AdaBoost/GA et filtrage particulaire: La vision par ordinateur au service de la sécurité routière

Yotam Abramson

ECOLE DES MINES DE PARIS
Collège doctoral

THESE

pour obtenir le grade de
Docteur de l'Ecole des Mines de Paris
Spécialité Informatique-Robotique

Présentée et soutenue publiquement
par
Yotam ABRAMSON

le 7 décembre 2005

AdaBoost/GA et filtrage particulaire :La vision par ordinateur au service de la sécurité routière

Directeur de thèse: Claude LAURGEAU

Jury

Prof. Claude LAURGEAU, Ecole des Mines de Paris, Président
Prof. Patrick SIARRY, Université de Paris 12, Rapporteur
Prof. Laurent TRASSOUDAINE, Université Blaise Pascal (LASMEA), Rapporteur
M. Bruno STEUX, Ecole des Mines de Paris, Examinateur
Mme Marie-Line GALLENNE, LCPC, Examinateur
M. Sylvain MILLEMAN, PSA, Examinateur
M. Philippe DOYEN, VEOLIA Transport, Examinateur

Remerciements

Je ne suis pas un thésard conventionnel. Au début de l'année 2002, après 8 ans d'expérience professionnelle, j'ai décidé de ne pas suivre le chemin traditionnel de l'industrie high-tech israélienne, mais de consacrer mon temps à la recherche de solutions au problème des accidents de la route. Echanger à l'âge de 33 ans un salaire de 100,000 euros par an pour un statut de thésard n'a pas été une démarche simple. J'aimerais remercier les personnes qui m'ont permis d'y parvenir.

Tout d'abord, je tiens à remercier le Prof. Claude LAURGEAU, Directeur du Centre de Robotique à l'École des Mines de Paris. Lorsque j'ai quitté mon travail et recherché des opportunités pour me spécialiser dans le domaine des systèmes de transport intelligent (STI), j'étais particulièrement attiré par les activités que le Pr. LAURGEAU développe au Centre de Robotique. Dès mars 2002 et jusqu'au début de ma thèse en septembre 2002, le Pr. LAURGEAU m'a accueilli à plusieurs reprises afin de trouver la formule qui me conviendrait.

Ensuite je remercie ma femme, Tali IVRY-ABRAMSON, d'avoir tout laissé pour me suivre en France, de garder nos enfants, notre maison et de me permettre ainsi de suivre tranquillement mes études.

Et enfin je remercie ma mère Zivit ABRAMSON et ma belle mère Yardena IVRY, de nous avoir soutenus pendant ces trois années.

Voici pour la partie matérielle. Et maintenant au côté professionnel.

Encore une fois merci au Pr. LAURGEAU de m'avoir ouvert la porte du monde des STI, de m'avoir permis de participer aux projets avec les partenaires industriels et académiques aux niveaux français et européen. Le modèle de laboratoire que le Pr. LAURGEAU a établi, en trouvant le bon équilibre entre recherche académique et résultats pratiques, sera toujours présent à mon esprit dans mon travail.

Je remercie également mon superviseur, le Dr. Bruno STEUX, pour les nombreuses heures qu'il m'a consacrés pour ma thèse et les projets industriels. Grâce à Bruno, j'ai beaucoup avancé dans le domaine de la vision artificielle.

Merci aux gens du Centre de Robotique, Fawzi NASHASHIBI, Bogdan STANCIULESCU, François GOULETTE ainsi que tous mes autres collègues (pardonnez-moi mais je ne peux pas mentionner tous les noms), pour le travail en équipe, pour des discussions fructueuses et vos remarques sur ma thèse.

Côté industriel, un remerciement spécial et tous mes vœux de succès à M. Christophe WAKIM de Renault pour notre agréable collaboration.

Many thanks to Dr. Yoav Freund from Columbia University (now in UC San-Diego) for our fruitful and enjoyable collaboration. It was an honor for me to work with a scientist of a

world-class reputation like Yoav.

Special thanks to Dr. Wido Kruijtzer from Philips Eindhoven for his support in the distribution of the SEVILLE idea and in the CAMELLIA project in general.

Je terminerai avec deux ultimes remerciements : à nos amies en France, Charlotte TETREL et Simone PORGES, pour leur aide pendant notre séjour, et à mes enfants, Itamar et Nadav. C'est pour eux que j'ai choisi de m'engager dans la lutte contre les accidents de la route.

Publications dans le cadre de cette thèse

- A. Khammari, F. Nashashibi, Y. Abramson and C. Laurgeau, "Vehicle Detection Combining Gradient Analysis and AdaBoost Classification", in ITSC '05, September 2005, Vienna, Austria.
- Yotam Abramson, Bruno Steux and Hicham Ghorayeb, "Yet-Even-Faster real time object detection", in ALART '05, September 2005, Siegen, Germany.
- Yotam Abramson and Yoav Freund, "SEmi-automatic VIsuaL LEarning (SEVILLE) : Tutorial on Active Learning for Visual Object Recognition", in CVPR '05, June 2005, San-Diego, California.
- Y. Abramson and B. Steux, "Hardware-friendly detection of pedestrians from an on-board camera", in IV '04, June 2004, Parma, Italy.
- Y. Abramson and B. Steux, "Robust real-time on-board vehicle tracking system using particle filtering", in IAV '04, July 2004, Lisbon, Portugal.

Résumé

Cette thèse combine des résultats récents et des algorithmes originaux pour créer deux applications temps-réel robustes d'aide à la conduite (projet aussi appelé "le véhicule intelligent"). Les applications - commande de croisière adaptative ("ACC") et prédiction d'impact piétons - sont conçues pour être installées sur un véhicule et détectent d'autres utilisateurs de la route, en utilisant une seule caméra frontale.

La thèse commence par un état de l'art sur la vision artificielle. Elle s'ouvre en passant en revue certaines avancées récentes dans le domaine. En particulier, nous traitons l'utilisation récente d'un nouvel algorithme nommé AdaBoost pour la détection des objets visuels dans une image, rapidement et sûrement. Nous développons la théorie, ajoutons des algorithmes et des méthodes (y compris une variante à base d'algorithme génétique) et en améliorons les résultats, dans le but d'adapter ces algorithmes aux besoins de vraies applications d'aide à la conduite. En particulier, nous prouvons plusieurs nouveaux résultats sur le fonctionnement de l'algorithme AdaBoost.

Toujours sur le plan théorique, nous traitons des algorithmes d'évaluation de mouvement et des filtres particulaires et leur utilisation dans la vision. De ces développements algorithmiques, nous arrivons à la description de deux applications d'aide à la conduite, toutes les deux entièrement mises en application, validées et démontrées sur le véhicule d'essai du Centre de Robotique de l'Ecole de Mines de Paris.

La première application, la commande de croisière adaptative ("ACC"), exploite les formes caractéristiques des véhicules pour les détecter. Ainsi, l'application détecte des véhicules en utilisant un ensemble d'algorithmes classiques de traitement d'image (détection d'ombres, des feux arrières, de symétrie et de bords), ainsi que l'algorithme AdaBoost mentionné ci-dessus. Cet ensemble d'algorithmes est fusionné en se plaçant dans le cadre du filtrage particulaire, afin de détecter les véhicules devant notre voiture. Puis, le système de contrôle prend pour cible la voiture située devant et garde une distance constante par rapport à celle-ci, tout en commandant l'accélération et le freinage de notre voiture.

La deuxième application, la prédiction d'impact piétons, estime au temps t la probabilité d'un impact de notre voiture avec un piéton au temps $t+\delta$. Dans l'application, la trajectoire de chaque piéton est calculée, et la probabilité d'impact est calculée selon la direction du piéton, du bruit et d'autres facteurs. Cette application utilise un accélérateur matériel spécifique créé dans le cadre du projet européen CAMELLIA (Core for Ambient and Intelligent Imaging Applications).

Mots-clés : vision artificielle, systèmes de transport intelligent, détection de piétons, commande de croisière adaptative.

This thesis presents two ITS applications, which are designed to be installed on a moving vehicle and detect other road users, using a single frontal camera. The two applications are Stop&Go ACC and Pedestrian impact prediction.

The thesis opens by describing the history and current status of the ITS domain. We review several existing systems which represent several approaches and research directions. Among these systems there are ones which are operational or almost operational, and ones which are futuristic.

Next we present some novel results in the field of computer vision/machine learning. These results are using, and are partly motivated by, the example of pedestrian detection. In particular we present new type of weak-classifiers to be learned by the AdaBoost algorithm, a classifier which is working faster than others and is not dependant of scene lighting conditions. We also present a novel way to collect large high-quality training sets in order to vastly improve the training results.

Using these results, we present a Stop&Go adaptive cruise control (ACC). We implemented this application with a set of known image processing algorithms, demonstrating how the combination of several relatively-simple algorithms can yield a reliable system. The application is running in 10 images per second and follows the car in front, while using a motion estimator to detect cut-ins.

Our second application is a pedestrian detection and impact prediction application. The system is running in 10 image per second and reliably predict the probability of an impact with a pedestrian in some time frame.

Keywords : computer vision, intelligent transportation systems, stop&go ACC, pedestrian detection.

Table des matières

Chapitre 1

Introduction

1.1 État de l'art

Les transports se sont considérablement développés au siècle dernier. Aujourd'hui, on peut atteindre n'importe laquelle des régions habitées dans le monde dans un temps raisonnable. Au début du 20ème siècle, les gens n'imaginaient pas qu'une personne aille quotidiennement travailler à 30 km de chez elle. De nos jours, cette situation est très commune.

Cependant, malgré cet important développement, de sérieux problèmes persistent. Les perturbations du trafic et les accidents sont sans doute les plus courants. Ces problèmes sont survenus lorsque la production en masse de voiture a débuté.

Aujourd'hui, environ 700 000 personnes sont tuées et 10 millions sont blessés chaque année dans le monde lors d'accidents de voiture, des milliards d'heures sont perdues dans les embouteillages et la recherche de places de parking. Par exemple, en Californie, les embouteillages coûtent 21 milliards de dollars chaque année en temps perdu et en gaspillage de carburant.

Il existe quelques solutions, partielles et onéreuses à ces problèmes. La première est l'usage des transports en commun. Les problèmes évoqués ci-dessus (ou bien la plupart d'entre eux) n'existent pas dans certaines catégories de transports tels que les trains. Cependant, la voiture personnelle demeure aujourd'hui le seul moyen de transport porte-à-porte et, par conséquent, ne pourrait être totalement remplacée. La seconde solution est de construire davantage d'infrastructures - autoroutes, barrières de sécurité et d'autres moyens de prévenir les accidents. C'est une solution très coûteuse, et donc qui ne répond au problème que partiellement.

Les "Systèmes de Transport Intelligent", ou STI, sont un domaine de recherche qui a pour but d'associer des systèmes informatiques à l'infrastructure du trafic et aux véhicules afin d'en améliorer les performances. Les applications "STI" sont conçues pour être installés dans les voitures et/ou sur l'infrastructure du réseau routier afin de prévenir des accidents et des perturbations du trafic.

Un état de l'art des STI est présenté au chapitre 2. Cette introduction propose seulement un bref aperçu du domaine.

Au premier niveau des applications des STI se trouvent les systèmes de gestion du trafic basés sur l'infrastructure. Ces systèmes permettent de renseigner le conducteur sur la vitesse à laquelle il doit rouler, l'informe des incidents et des embouteillages. Ces systèmes sont déjà

sur le marché.

A un niveau plus avancé, les STI commencent à pénétrer les véhicules. L'idée est d'équiper un véhicule avec des capteurs, afin qu'il puisse "comprendre" son environnement. Cette "compréhension" peut être utile pour informer le conducteur ou même s'y substituer en cas de défaillance. Dans cette famille de produits, encore à l'état de prototype, on trouve des applications d'avertissement de collision à l'avant (*FCW - Forward Collision Warning*) et de régulation de vitesse automatique (*ACC - Adaptive Cruise Control*).

La vision finale de la communauté STI est, naturellement, des voitures qui circulent toutes seules. Une personne pourrait entrer dans sa voiture, entrer la destination désirée et y arriver sans intervention humaine. Cela reste naturellement une vision très lointaine. Un premier essai pour mettre en application un tel système fut les "systèmes autoroutier automatisés" (*AHS - Automated Highway Systems*) décrits dans le chapitre 2. Le succès des expérimentations techniques ne fut pas suivi d'un succès industriel : ces systèmes ne sont pas installés aujourd'hui sur de vraies autoroutes et ne sont pas prêt de l'être. Il y a de cette expérience de nombreux enseignements à tirer quant à l'acceptabilité par le public de systèmes complexes où des machines prennent un contrôle jusqu'alors dévolu à l'être humain.

Dans cette thèse, nous traitons du sous-domaine des systèmes installés à bord des véhicules - les rendant ainsi "intelligents" -, et en particulier de systèmes basés sur des caméras. Les caméras sont des capteurs très peu onéreux, à l'information très riche - notamment du fait de leur large ouverture -, sans partie mécanique fragile. Leur principal avantage est que la conduite humaine est basée elle-même essentiellement sur la vue, et donc l'environnement à observer - la route, et les autres véhicules - est naturellement très structuré. Comme les algorithmes de vision progressent continûment, de plus en plus d'information peut être extraite à partir des caméras.

Des travaux dans le domaine sont menés dans le monde entier, dans les divisions de recherche de beaucoup de fabricants de voitures ou d'équipementiers, ainsi que dans les écoles et universités. Une vue d'ensemble de ces activités est donnée dans le chapitre 2.

Dans notre laboratoire au Centre de Robotique à École des Mines de Paris, nous travaillons sur diverses applications du véhicule intelligent. Quelques exemples des travaux incluent :

- La fusion entre radar et vision, en vue d'exploiter les avantages des différents capteurs suivant les situations.
- Le contrôle latéral. Ce système contrôle automatiquement le volant du véhicule en fonction de la position du véhicule sur la route mesuré par la caméra embarquée..
- La détection de piétons, afin d'alerter le conducteur.
- Le capteur de trafic. C'est un système qui mesure le flot de circulation au lieu de la voiture à un moment donné. La mesure est faite en combinant l'observation par la caméra et les données présentes sur le bus CAN du véhicule (vitesse, clignotants, etc.).
- L'avertissement de collision frontale. C'est un système qui alerte le conducteur quand il y a un risque de choc avec la voiture de devant.

Ces exemples sont élaborés dans le chapitre 2. Mon travail a été concentré sur deux applications :

- La commande de croisière adaptative (*ACC - Adaptive Cruise Control*).
- La détection de piétons

Ces deux applications sont décrites dans la prochaine sous-section. Tout en mettant en

œuvre ces applications, j'ai développé quelques idées novatrices dans le domaine de la détection visuelle d'objets. Ces résultats forment le fond théorique de ma thèse.

1.2 Description des applications

Le premier problème traité dans cette thèse est le problème de la commande longitudinale à vitesse réduite de la voiture. Dans la circulation dense, le conducteur doit accélérer et freiner très fréquemment. Ceci accélère l'usure de la voiture, augmente la consommation et peut également causer des accidents avant-arrière. Ce genre d'accident, bien qu'habituellement non mortel, est très fréquent et cause de nombreux embouteillages.

Afin de résoudre le problème de la conduite en circulation dense, un système appelé la commande de croisière adaptative (*ACC - Adaptive Cruise Control*) peut être développé. Considérant que la "commande de croisière" traditionnelle garde une vitesse constante, sa version adaptative garde elle une distance constante à la voiture de devant. De telles applications existent déjà aujourd'hui sur les voitures de haut de gamme et utilisent le radar pour détecter la voiture de devant. Ces systèmes d'ACC sont très chers et relativement peu fiables, du fait des faibles angles d'ouverture des radars utilisés et donc de leur inefficacité en courbe.

Ici, nous avons développé un système qui est basé sur la vision et par conséquent peut être installé sur des voitures de bas de gamme. Ce système cependant est conçu pour fonctionner à vitesse réduite, c'est à dire dans une circulation dense, quand le conducteur s'arrête et repart sans arrêt. C'est pourquoi il porte ici le nom d'application "Stop'n'Go".

* * *

Les chocs avec des piétons expliquent approximativement 25% des accidents mortels et 17% des dommages liés à des accidents dans le monde entier. Ces accidents sont provoqués par les conducteurs qui ne voient pas les piétons qui croisent leur trajectoire.

Un système installé dans la voiture qui détecterait les piétons serait donc utile. L'application présentée dans cette thèse, la détection de piétons et la prédiction de choc piéton, détecte les piétons situés devant notre voiture.

Une fois détecté, un piéton est localisé et suivi. Selon sa vitesse et sa direction, le système calcule la probabilité de l'impact entre notre voiture et ce piéton dans l'intervalle de temps qui suit. Si cette probabilité est supérieure à un seuil donné, une alerte au conducteur est déclenchée.

L'application est novatrice parce qu'elle obtient le taux le plus élevé obtenu jusqu'ici en matière de détection de piétons en temps réel. Des applications plus lentes existent, qui détectent des piétons avec un taux plus élevé de succès, mais elles ne sont pas adaptées à des applications en temps réel. En outre, c'est la première fois qu'une application calcule en temps réel la probabilité de l'impact avec un piéton. Une telle probabilité peut être employée également pour appliquer des mesures de sécurité, tel que le déclenchement d'un airbag piéton ou plus simplement l'anticipation du freinage. Une diminution même relativement faible de la vitesse de l'impact peut réduire considérablement les dégâts liés à l'impact.

1.3 Contribution théorique

Au cours du développement des deux applications, j'en suis arrivé au besoin de créer un système fiable et rapide de détection d'objets, qui détecterait efficacement des voitures et des piétons dans une image en temps réel. Le système que j'ai développé et qui est présenté dans cette thèse, est basé sur celui de Viola et Jones [Viol 02] et est amélioré en ajoutant un nouveau genre de primitives visuelles, discuté en détail dans le chapitre 3. Ces primitives permettent de détecter des voitures et des piétons plus rapidement et mieux que les méthodes existantes. Dans le domaine de la détection de piétons, nous montrons dans cette thèse que notre système obtient de meilleurs résultats que n'importe quel travail publié dans le domaine, en utilisant des images statiques en noir et blanc.

Dans le développement de l'application d'ACC, nous avons employé pour la première fois la méthode du filtrage particulaire, qui donne une localisation précise de la voiture de devant.

1.4 Plan de la thèse

Dans cette thèse, le chapitre 2 fournit un examen complet de l'historique et des développements courants dans le domaine des STI, dans le monde et dans notre laboratoire. Puis, il présente nos choix et justifie l'approche spécifique mise en œuvre dans nos applications.

Le chapitre 3 donne la situation actuelle dans le domaine de l'apprentissage automatique pour la détection visuelle d'objets, suivie de nos développements à la base de nos applications. Le même schéma est suivi dans le chapitre 4 et le chapitre 5, qui portent respectivement de la détection du mouvement et du filtre particulaire. Les applications elles-mêmes sont décrites dans les chapitres 6 et 7.

1.5 Background

Transportation systems have dramatically developed during the last 100 years. Today, one can arrive, within reasonable times, between almost any inhabited points on earth. In the beginning of the 20th century, people could not imagine a situation where a person moves 30 km from his home to work every day. Nowadays, such a situation is not rare.

However, despite of this great development, serious problems still exist. Traffic congestion and accidents are perhaps the most common ones. These problems emerged as the mass production of cars began in the world. Today, about 700,000 people are killed and 10 million are injured every year worldwide in traffic accidents, and billions of hours of time lost in traffic jams and search-driving for a parking lot (for example, only in California, traffic jams statewide cost $21 billion a year in lost time and wasted fuel).

There are some expensive, partial solutions to these problems. The first is public transportation. The above problems do not exist (or almost don't exist) in some forms of public transportation such as trains ; however, the private car remains today the only mean to provide door-to-door service and therefore cannot be fully replaced.

The second solution is to build more infrastructure - highways, security fences and other accident preventive infrastructure. This is a costly solution, and as such can address these

problems only partially.

Intelligent Transportation Systems (ITS) is a domain that tries to combine computerized systems in traffic infrastructure and vehicles in order to increase its performance. ITS are designed to be installed in cars and/or in road infrastructure in order to prevent accidents and traffic congestion.

A full review of the domain of ITS is given in chapter 2 ; in this introduction we will only provide a brief overview. In the "low level" ITS applications we can see the infrastructure-based traffic management systems, these systems that tell drivers about what speed to drive and inform them about incidents and traffic jams. This type of systems is already in the market today.

More advanced, ITS systems are beginning to enter the vehicles themselves. The idea is to equip the vehicle with sensors in order to allow it to "understand" its environment. This "understanding" can be used to inform the driver or to apply some operations. In this family of products, which is found today in development stages, we can find applications such as Forward Collision Warning (FCW) or Adaptive Cruise Control (ACC).

The ultimate vision of the ITS community is, naturally, cars that drive by themselves. A person could enter his car, enter the desired destination and get there without human intervention. This, of course, is a far vision. A first attempt to implement a similar system was in the case of the Automated Highway Systems (AHS) which is described in chapter 2. The technical success, together with the fact that these systems are *not* installed today in real roads, can teach us a bit about how ITS systems will propagate in the future.

In this thesis, we deal with the ITS sub-domain of in-vehicle smart systems, mostly ones which are based on cameras. Cameras are relatively cheap sensors which give broad view of the scene, and as vision algorithms advance, more information can be extracted from them. Work in this sub-domain is carried out in the research division of many car manufacturers, in other companies and in academic institutes worldwide. An overview of these activities is given in chapter 2.

In our laboratory in the Center of Robotics in École des Mines de Paris, we are working on various in-vehicle applications. Some examples of the works include :

- *Fusion of radar and vision* : ways to integrate the information coming from radar and video cameras.
- *Lateral control* : automatic control of the wheel of the vehicle by a video camera observing the lane markings.
- *Pedestrian detection* : a system that detects pedestrians using a frontal camera and alerts the driver.
- *Traffic sensor* : a system that knows the situation of the traffic congestion at the location of the car at a given moment. This is done using various sensors, including camera and information on the speed of the car.
- *Forward collision warning* : A system that alerts the driver when the headway to the car in front is too small.

These examples are elaborated in chapter 2.

My work was focused on two applications - adaptive cruise control and pedestrian detection, described in the next subsection. While implementing these applications, I developed some innovative ideas in visual object detection. These results form the theoretical back-

ground of my thesis.

1.6 Description of applications

The first problem addressed in this thesis is the problem of low-speed longitudinal control of the car. In heavy traffic, the driver has to accelerate and brake very frequently. This accelerates the erosion of the parts of the car and can cause front-back accidents. This kind of accidents, although usually not fatal, is still severe and can also cause heavy traffic jams.

In order to solve the problem of driving in heavy traffic, a system called "adaptive cruise control" can be developed. Whereas the traditional "cruise control" is keeping steady speed, its adaptive version keeps a steady headway to the car in front.

Such applications exist on high-end cars and are using radar to "lock" on the car in front. These ACC systems are very expensive. Here, for the first time, we have developed a system which is based on vision and hence can be installed on low-end cars. The system, however, is designed to work in low-speed, that is, in heavy traffic, when the driver stops and goes all the time. This is why it is usually referred as "Stop and go".

* * *

Pedestrian accidents account for approximately 25% of fatal accidents and 17% of injuries accidents worldwide. These accidents are caused by drivers who don't see the pedestrians entering their trajectory. A device installed in the car that can detect pedestrians can be therefore useful. The *Pedestrian detection and impact prediction* application presented in this thesis, is detecting pedestrians in front of our car. Once detected, a pedestrian is localized and tracked. According to his speed and direction, the system calculates the probability of impact between our car and that pedestrian within δ from now. If this probability is superior of a given threshold, an alert to the driver is issued.

The application is innovative because it obtains the highest rate published so far of pedestrian detection in real-time. Slower applications exist which detect pedestrians with higher success rate but they are not adequate for real-time applications. In addition, this is the first time where an application is calculating in real-time the probability of impact with a pedestrian. Such a probability can be used also for applying some safety measurements.

1.7 Theoretical contribution

During the development of both applications, I have come to a need to create a reliable and fast object detector scheme, one which will reliably detect cars and pedestrians in an image in real-time. The scheme I developed, which is presented in this thesis, is based on the one of Viola and Jones [Viol 02] and is improved by adding new kind of visual features, discussed in details in chapter 3. These features are allowing to detect cars and pedestrians faster and better than the existing methods. In the domain of pedestrian detection, we show in this thesis that our system obtains better results than any published work in the field, using black and white static images.

In the development of the ACC application we have used for the first time the method of particles filter, which give steadier localization of the car in front.

1.8 Program of this thesis

In this thesis, chapter 2 is providing a thorough review of the ITS history and current developments, in the world and in our laboratory. Then it explains the motivation of choosing the specific approach we took to implement the applications.

Chapter 3 gives the state of the art in machine learning for visual object detection, followed by our developments that form the basis to the implementation of the applications. The same format is used in chapters 4 and 5 that deal with motion estimation and particles filter, respectively.

The applications themselves are described in chapters 6 and 7.

Conclusions are given in chapters 8.

Chapitre 2

Intelligent Transportation Systems (ITS)

2.1 What is ITS ?

2.1.1 Introduction

Intelligent Transportation Systems, or ITS, is a general name for every computerized system integrated into the transportation infrastructure or vehicles. Such systems help to reduce accidents and trafic congestion by helping the driver to drive and, later on, automating some parts of the driving process.

2.1.2 Historical background

ITS Timeline

History of ITS can be divided [Hide 96] into 3 stages :

Stage 1 - The beginning of ITS research - stretches back to the 1960s and 1970s. This early research was characterized by Japan's CACS, the Electronic Route Guidance System (ERGS) in the United States, and a similar system in Germany. All of these systems focused on route guidance and were based on central processing systems with huge computers and communications systems. These systems never resulted in practical application because of the limitations of computers of that era.

The next period, from 1980 through about 1995, could be called stage 2. Conditions for ITS development had improved by the 1980s. Improvements in computers - memory and computation power - made applications possible. New research and development efforts started :

- In Europe two projects were going on : the PROgraM for a European Traffic System with Higher Efficiency and Unprecedented Safety (PROMETHEUS), which was established by car manufacturers, and the Dedicated Road Infrastructure for Vehicle safety in Europe (DRIVE), organized by the European Community.
- In the United States, there was the Intelligent Vehicle-Highway Systems (IVHS) project.
- In Japan, work on the Road/Automobile Communication System (RACS) project, began in 1984.

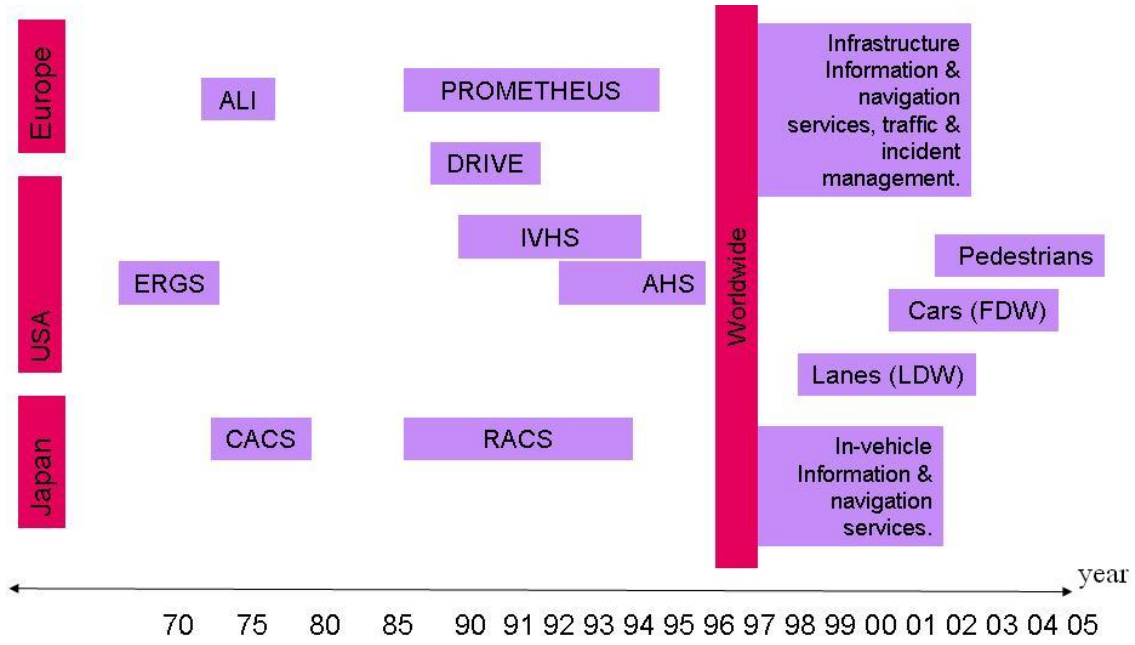


FIG. 2.1 – History of ITS development : in the early and middle period of ITS development, large projects are presented, divided by geographical areas. In the last 10 years, the numerous individual projects are not presented, but the domains in which worldwide research and development were carried out.

Our situation today can be considered as Stage 3. In this stage practical applications are starting to appear ; the potential of ITS is fully understood, but also the ways in which it has to be implemented. We are more aware of deployment problems and social and economical limitations.

Figure 2.1 gives a timeline of the development of ITS.

2.1.3 ITS Categories

When speaking about intelligent transportation systems one can distinguish between three kinds of systems :

1. Systems installed in the transportation infrastructure, that is, on the road, without assisting any system components installed on cars.
2. Systems installed on a car and functioning independently, that is, without assisting any system components outside of the car.
3. Systems which consist on components which are installed both on a car and on the infrastructures. These systems employ cooperation between the mobile and immobile component of the system.

Nowadays, the overwhelming majority of the operational ITS belongs to type 1. This type of systems can be described as a high tech extension of the existing infrastructure (roads, bridges). They can tolerate relatively high cost, because one system is serving many cars.

They are initiated and created mostly by public authorities and do not suffer from deployment problems. Subsection 2.2 provides a detailed description of these systems.

Systems of type 2 above are said to be "the next step" in ITS. Some of these systems are already integrated in today high edge vehicles, and the majority will be made available in the next decade. This type of systems suffer from severe limitation of price, since the price of such a system is added to the cost of each individual car. As a result, most popular systems will use very cheap sensors like simple cameras and lidars. This type of systems is described in subsection 2.3.

Systems of type 3 were developed and tested in the last decade, with the American AHS being the most standing example. The big advantage of these systems is that they allow almost any function to be realized : once an on board system has the cooperation of an outside system (e.g. road beacons with radio connection) it can provide excellent results, such as totally automatic driving in any weather condition. One can consider the control of airplanes as an example of what can be achieved with such cooperative systems. However, as was slowly discovered during the period of these developments, these technologies suffer from severe problems of deployment, even before arriving to the question of system price. The deployment problem means, essentially, that it is not clear who will install the system first : will it be the car manufacturers who will install expensive equipment in their cars when the infrastructure part is still not active, or will it be the authorities (e.g. highway managers) which will install expensive equipment and possibly cause inconvenience to the general public before having the majority of the cars equipped with the necessary equipment ? The situation, as of today, is that none of these poles is going to make the first move. This is discussed in subsection 2.4.

2.2 Infrastructure-based ITS

2.2.1 Informative infrastructure systems

Infrastructure ITS applications are the most popular ones today. This type of applications are usually carried out by state authorities using the same procedure by which roads are constructed ; they don't need any cooperation with car manufacturers.

Infrastructure ITS applications mostly give information to drivers. They can relieve traffic congestions or alert from dangers. Here are some common types of such applications :

- *Advanced Traffic Management Systems (ATMS)* employ a variety of relatively inexpensive detectors, cameras, and communication systems to monitor traffic, optimize signal timings and thus control the flow of traffic. For example, we see in figure 2.2 a traffic sign in Paris which informs the drivers about the time it takes to arrive to a certain point.
- *Incident Management Systems*, for their part, provide traffic operators with the tools to allow quick and efficient response to accidents, hazardous spills, and other emergencies.
- *Electronic toll systems* are implemented in toll-roads and allow easy registration of the cars passing in the road. An example to this type of applications is found in Israel's road number 6 (the Yitzhak Rabin Cross Israel Highway), employing ITS systems made by

FIG. 2.2 – A traffic sign in France, telling the number of minutes it takes to arrive to a certain point

FIG. 2.3 – Cross-Israel highway, with cameras that identify the plate number of the passing cars.

Raytheon. The system provides fully automatic toll collection, either by cameras which "read" the license plates of the passing cars or by contacting a special transponder that frequent drivers are encouraged to buy. In both cases, drivers do not have to stop in the entrance to the toll-road ; their credit cards are being charged automatically (or they receive the bill via mail if no credit card information is available).
Raytheon's system in Israel's road number 6 employs for the first time several advanced features, such as real-time enforcements of speed-limits and the use of transponder data for incident detection.

2.2.2 The smart junction

The idea of the so-called "smart junction" is that the junction will detect dangerous situations and alert drivers before an accident occurs. Following is one important example.

Research on infrastructure-based systems is currently being conducted under the Infrastructure Consortium, which is comprised of the U.S. DOT, and the California, Minnesota, and Virginia DOTs, which are sponsoring the Intersection Decision Support System research project. The research is being conducted by University of California at Berkeley PATH (Part-

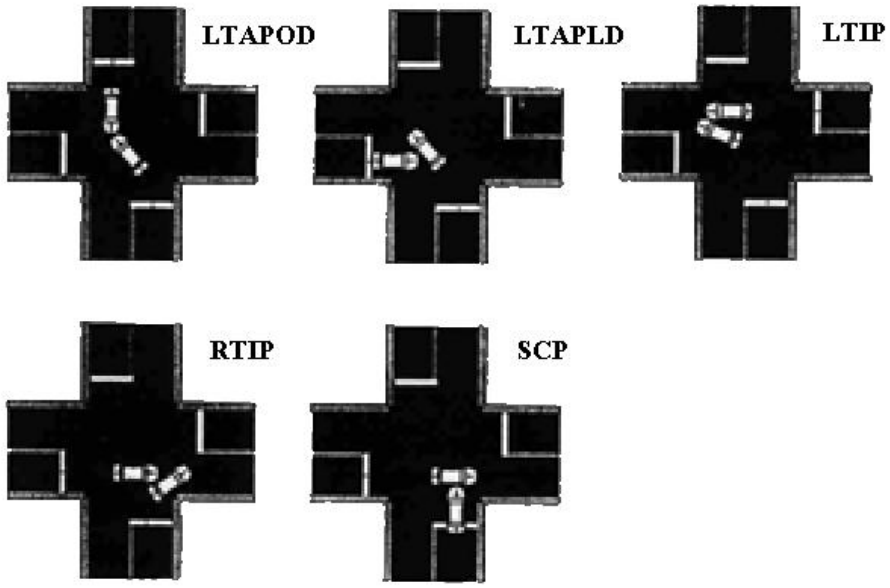


FIG. 2.4 – Intersection Collision Scenarios. **Top, from left to right** : Left Turn Across Path Opposite Direction, Left Turn Across Path Lateral Direction, Left Turn Into Path. **Bottom, from left to right** : Right Turn Into Path, Straight Crossing Path

ners for Advanced Transit and Highways) Program, University of Minnesota Intelligent Transportation Systems Institute, and Virginia Polytechnic Institute/Virginia Tech Transportation Institute.

Intersection Collision Avoidance Systems use sensors to gather information about vehicle movements near an intersection, process that information to determine if a collision is at risk of occurring, and issue warnings to drivers of vehicles in danger. They differ from traffic control signals in that they are continuously processing information when vehicles are present and creating messages tailored to specific vehicles' paths, speeds and driver behaviors.

Intersection crashes may be classified into one of the following five categories, as shown in Figure 2.4 :

a Left Turn Across Path - Opposite Direction

b Left Turn Across Path - Lateral Direction

c Left Turn Into Path

d Right Turn Into Path

e Straight Crossing Path

Roughly 43 percent of vehicle crashes in the U.S. occur at intersections or are intersection-related. A significant share of them take place at intersections with traffic signals or stop signs. Their causes are often due to drivers' misjudgment of the situation, failure to correctly observe the situation, or inability to accurately perceive the degree of danger. Some 60% of rural intersection crashes occur even after the driver entering the main roadway has stopped before proceeding, researchers have found. Another significant share of crashes are caused by

FIG. 2.5 – Left Turn Across Path at UC Berkeley : note the red sign that alerts from dangers.

the driver entering the intersection against the signal or failing to stop at a stop sign. These findings suggest that interventions such as warning systems and driver assistance could be particularly effective in reducing intersection crashes.

Left Turn Across Path : Opposite Direction at UC Berkeley PATH

Opposite Direction crashes account for 27.3% of intersection related crashes in the US. Two-thirds of them occur at signalized intersections. Reasons for these types of crashes include :

- Failure to judge safe gaps in traffic correctly,
- Failure to judge speeds of closing vehicles correctly,
- Obstruction of driver's view,
- Failure to perceive opposing vehicle.

The PATH research is working with remote sensors for upstream traffic and loop detectors to detect downstream traffic and the subject vehicle. Sensors measure and relay range, rate and trajectories. The sensor data is processed in a warning algorithm. If a potential conflict is detected, a signal is sent to a dynamic warning sign, which will activate (see figure 2.5).

Left Turn Across Path : Lateral Direction at University of Minnesota ITS Institute

University of Minnesota's ITS Institute demonstrated an IDS system that addressed Left Turn Across Path : Lateral Direction crashes, which typically occur in rural areas when a vehicle attempts to cross or turn onto a road at an unsignalized intersection. ITS Institute

researchers found that 60 percent of crashes at rural intersections happen even after drivers stop before proceeding into the intersection. They termed this a gap perception problem.

The ITS Institute system was designed to tell a driver if it is unsafe to enter the main roadway. Radar detectors are deployed at five points around the roadway to detect approaching vehicles. The detectors communicate to a central processor via a wireless connection. The processor then runs an algorithm, which calculates which gaps are safe or unsafe to enter the roadway. Depending on the result, the algorithm may activate an LED no-left-turn signal.

Straight Crossing Path Crashes at Virginia Tech Transportation Institute

According to Virginia Tech Transportation Institute researchers, approximately 30% of intersection crashes involve vehicles executing a straight crossing path. The Institute research is focusing in ways to prevent those crashes caused by traffic signal and stop sign violations.

The system being tested includes pole-mounted radar at signalized intersections to determine an approaching vehicle's speed and location and warn the driver with dynamic signing (LED stop sign and strobe light) if a violation is likely. An additional countermeasure that researchers have begun is a set of "intelligent" rumble strips that would deploy if a violation seemed imminent.

2.3 IV independent applications

Under the definition of Intelligent Vehicle (IV) independent applications we group all the applications which are not cooperating with any infrastructure device. These application therefore have to "get on by themselves" - find all the information they need through independent sensors such as camera, lidar or radar. In this section we will review some of the common applications - GPS-based localization systems (which barely passes the definition of "intelligent" system), as well as more advanced applications such as pedestrians detection, automatic cruise control (ACC), side and forward collision warning (SCW and FCW) and lane departure warning (LDW). A small part of these applications is already available in the market, mainly radar applications in high-end cars. More applications are expected to penetrate the market in the next five years. It is likely that this kind of applications will become a standard security device like the airbag.

2.3.1 On-board GPS-based localization system

The advances of global positioning systems (GPS) allowed most new cars today to be equipped with a system that give extra data to the driver. The GPS localization, together with a details map (which is pre-installed on the car's computer) can create systems that give driving directions to a desired destination ("never lost"). See figure 2.6.

When combined with traffic information (which can be downloaded from the internet), the system can calculate travel times and advice the driver as to what is the best route to a given destination.

FIG. 2.6 – An on-board never-lost system.

2.3.2 On-board signalization

The information found in road signs, road geometries and other infrastructure such as crosswalks, is huge. Today, each driver perceives the road signs visually and use them to control his vehicle. Many accidents are related to disrespect of road signs, resulting from human errors - tiredness, careless driving or bad interpretation. We can ask the following question : why can't technology today allow the vehicle to read the road signs automatically and report them to the driver ?

In the Center of Robotics in École des Mines de Paris, a system called "on-board signalization" was developed. The system makes it possible to perceive the road signs and to visually present them on the dashboard of the car.

We can imagine two technical approaches to solve the problem. The first consists in establishing a communication of the infrastructure towards the vehicle. The traffic signs becoming of the coded active transmitters, inform the vehicle of their presence. This solution is conceivable technically but not very realistic from an economic standpoint because it supposes any road's infrastructure is equipped. In France, for example, there are several million panels on the road network.

The second solution, which was chosen by the Center of Robotics, consists in exploiting the potential GPS systems (or soon Galileo) and of the numerical cartography (GIS systems).

France has a road network of approximately one million kilometers. Let us imagine that the every two hundred meters, one puts a point of marking on the roads with longitude, latitude and altitude coded in sixteen bits ; that makes for six bytes per point, thirty bytes per kilometer and a total of 30 megabytes to store all the roadmap of France. This is not a lot ; one can put all this data, together with the roads signs in these roads, in a single data CD.

Naturally, the production of such a database and its maintenance requires a considerable work of collection and verification of information. But the data already exist within the databases of the road network managers.

To demonstrate the system, a system was shown working in a test circuit. In figure 2.7 we can see several examples.

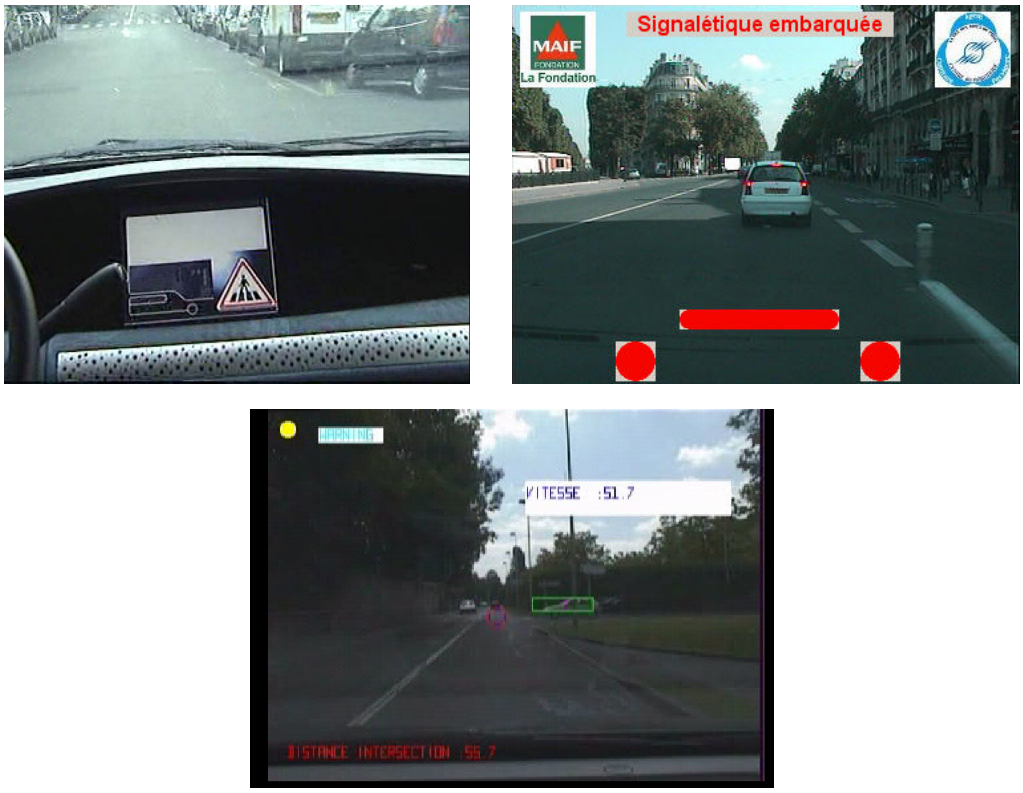


FIG. 2.7 – The on-board signalization of the Center of Robotics in École des Mines de Paris. Top left : the driver is alerted about the existence of a crosswalk. Top right : the driver is informed that the car in front is braking. Bottom : the driver is informed about the maximum allowed speed and that a vehicle is about to cross the junction.

2.3.3 Pedestrian detection

Applications of visual pedestrian detection are detecting the location of pedestrians in the environment of our vehicle and possibly, upon detecting an hazardous case, alert the driver or apply some safety measurements. Following there are two principal types of such an application.

Pedestrian visualization

The simplest application is presenting the driver with a map of pedestrians which are found in the environment of the car. The system does not do anything in case of potential danger, because it does not assume that its level of detection is exact enough in order to calculate reliable impact probability. Rather than that, pedestrians are simply presented to the driver, in case he didn't see them himself. This application can be very useful at night where the camera can use night vision and easily "see" better than the driver.

Once pedestrians in the area of the car have been detected, they are presented to the driver as red rectangles drawn on an on-board monitor, which exists already today in many new cars. Once certain types of visualization systems will be ready, the red rectangles could

FIG. 2.8 – Pedestrian visualization through the front glass

be drawn on the front glass of the car, as shown in figure 2.8. A camera installed in the interior of the car is detecting the position of the eyes of the driver. This position is used to calculate an exact location on the front glass where the red rectangle has to be drawn. The driver's attention is immediately drawn to the existence of the pedestrian.

The exactness of the detection needed for such application is not high. A rectangle which is located approximately on the pedestrian will do for this matter. Also, the false detection rate can be slightly higher than in other applications.

Pedestrian impact prediction

This type of applications will not just detect pedestrians in the surrounding of the car, but also try to estimate the probability of impact with our vehicle within a certain time interval (e.g. at time t the system is giving the probability of impact with *any* pedestrian at time $t + \delta$). In order to do that, the detection of the pedestrians has to be exact. Exact location of the legs and head must be known in order to calculate - over several images - the trajectory of the pedestrian, and to calculate the probability that it will intersect with the proximity of our car.

Once the impact probability is higher than a certain threshold, the system can alert the driver by drawing a red rectangle around the pedestrian (using a visualization system as described above) and/or making a sound. In a more advanced application, the system could automatically apply some active measurements like applying the brakes, opening an airbag or leveraging the front cover (see figure 2.9). Of course, the reliability level of such a system should be much higher in this case : while a driver might tolerate a small visual false alert once per week, he could definitely not tolerate a false application of the brakes even once per year. For this reason, fully automatic systems are still far from being implemented. Simple systems which alert visually and/or with a sound are expected in the coming years.

2.3.4 Lane departure warning (LDW)

Lane departure warning is one of the vision applications which are more likely to penetrate into passenger cars in the short term. It is relatively simpler than cars detection, and definitely from pedestrian detection.

One of the most advanced LDW systems which are already in advanced tests by cars manufacturers is the system of the Israeli company Mobileye [MobilEye]. Mobileye's Lane

FIG. 2.9 – Front cover which is opening to reduce pedestrian impact

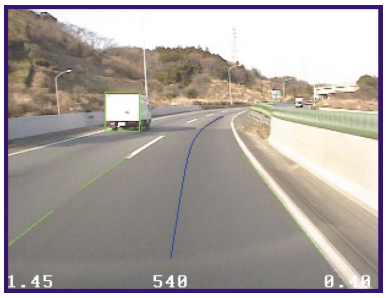


FIG. 2.10 – The lane departure warning system of MobilEye (source : Mobileye's website)

Departure Warning (LDW) system uses monocular image processing for detection of lanes marking on the road, and for measuring the position of the vehicle relative to the lanes. The LDW application provides indications to unintentional roadway departure.

According to the published information, Mobileye's system operates at all vehicle speeds and for all types of roads : highways, country roads and urban freeways. It is said to "perform well even in urban conditions" ; apparently urban conditions were found to be harder than non-urban ones, and this is obvious, due to the cluttered background and the high volume of cars inside the city. Mobileye's system is said to be able to detect and differentiate between different types of lane markings : solid, dashed, boxed and cat-eyes, and is not sensitive to the line width. In the absence of lane markings the system is claimed to utilize road edges (boundary between paved surface and ground) and curbs. It will be noted that this is an important feature of an LDW system, since many ways do not contain lines, but still much of the visual information hides in the environment.

Mobileye explains that the system fits a three-parameter road model that accounts for lateral position, slope and curvature. The curvature parameter is used for increasing the warning reliability under curved roads and for estimating time to lane crossing. In addition the system holds several lane models at all times so that it can choose between them immediately in ambiguous conditions such as urban roads, merging lanes, or exit lane situations.

As expected, the LDW software includes a "light" version of Mobileye's vehicle detection algorithm in order to avoid confusion of passing vehicles and their shadows with lane markings. Apparently, an LDW system which does not take into account the obstacles on the road can never function perfectly.

Mobileye's LDW system is claimed to incorporate an advanced warning scheme that

supports early warning capability, based on measurement of lateral vehicle motion, to predict the time to lane crossing providing an early warning signal before the vehicle actually crosses the lane.

The system is also said to take into account driver behavior and suppresses the warning signal accordingly. Is seems that such a system should indeed stop the alarm signal if the lane departure was suppressed, and not force the driver to shut off the alarm manually. In Mobileye's publications it is said that lane departure warnings are suppressed in cases of :

- Intentional lane departures (indicated by the turn signal on),
- No lane markings (e.g. within junctions),
- Inconsistent lane markings (e.g. road construction areas)

It will be noted that Mobileye's system is purely vision based. A good system of this type must sufficiently operate in a variety of weather and illumination conditions, to perform well under partially visible or poor lane markings, and in difficult conditions such as strong shadows, clutter and inclement weather conditions including rain and wet roads. When lane marks are not available or under poor visibility conditions, the system must be able to shut off and notify the driver.

Observing this information of Mobileye about the system, one can find out what are the characteristics of a good LDW system. This is correct regardless of whether Mobileye's systems functions indeed as described (it will be noted that the descriptions are probably not far from reality since the company has many evaluation agreements in advanced stages, and is expected to see its system on the road as early as 2007).

Enhancements of LDW

Mobileye offers its system with several possible "mini-applications", some of the comfort, some can enhance safety. These applications are nice ideas that demonstrate to us how in-vehicle intelligent applications are actually a "package" of driver support and not a set of separate applications.

The enhancements are :

- Lane keeping and heading control - slightly "helping" the driver to control the wheel *before* a lane departure occurs.
- Road geometry prediction - "seeing" the curves ahead and alerting the driver.
- Lane position monitoring for transportation fleets - something like the PROMETHEUS European project [G Re 95].
- Automatic headlamps activation.
- Intelligent high-beam control - when a car appears in front, can turn off high-beam until this car passes.
- Detecting wet road conditions and advising the driver to reduce speed.

2.3.5 Forward Collision Warning (FCW)

In order to review Forward collision warning we found it again useful to resort to Mobileye's publications. Their description of the system forms a good example of the characteristics of such a system (again, regardless if their description is correct or not).

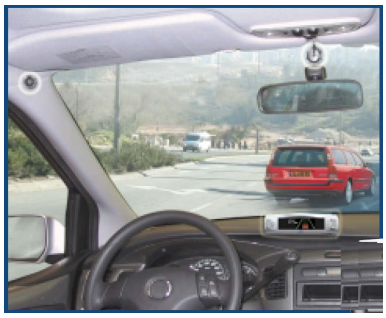


FIG. 2.11 – The forward collision warning of MobilEye (source : Mobileye's website)

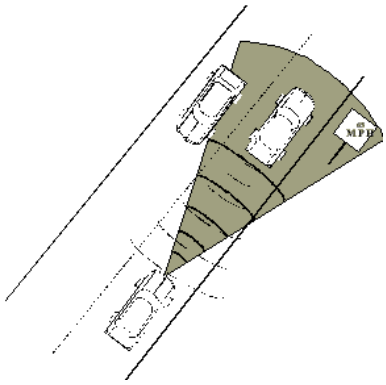


FIG. 2.12 – Adaptive cruise control.

Typically, a FCW system continuously computes time-to-contact to the vehicle ahead, based on range and relative velocity measurements. According to Mobileye's publications, their system uses an image processing algorithm to determine whether the vehicle ahead is in a collision path (even in the absence of lane markings). In case of collision conditions, the system provides audiovisual warnings to the driver at predetermined time intervals prior to collision (e.g. 2.5, 1.6 and 0.7 seconds, in increasingly stronger sounds) alerting the driver about the danger and allowing appropriate action such as braking or steering away from the obstacle ahead.

An important issue, as in the case of LDW, is the suppression of the alert. In Mobileye's case, it is said that the system uses information about driver actions (e.g. braking) to suppress warnings in situations that are under the driver's control.

2.3.6 Adaptive cruise control (ACC)

Adaptive cruise control improves on traditional cruise control by allowing a vehicle to automatically adapt to the speed of highway traffic - follow another vehicle at a set distance. With ACC, the driver selects a desired interval to follow traffic as well as the desired cruise speed.

When slower traffic is encountered the ACC alters vehicle speed to maintain the desired interval while following traffic. Speed is controlled by ACC with braking when needed. When

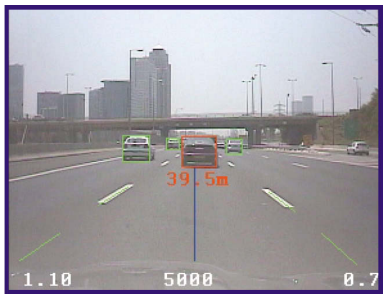


FIG. 2.13 – The Adaptive Cruise Control system of MobilEye (source : Mobileye's website)

traffic clears, ACC resumes the desired cruise speed. Of course, the driver may override the system by braking at any time.

How it works

When activated by the driver, a radar, laser and/or a video camera on the front of the vehicle determines the distance and relative speed of any vehicle which is in the path of driving. The ACC computer continually controls the throttle and brakes to maintain cruise speed or adapted speed of traffic.

ACC with radars exist today on the market on high-end cars (e.g. Mercedes-Benz S-class). ACC with vision and/or laser are on mature phases of development and are expected on the market towards 2007. One of the systems which is perhaps in most advanced stages of deployment is, again, the Mobileye system, and we chose to review it briefly here.

The MobilEye ACC

As in other systems, in Mobileye's system the camera detects and classifies targets in front of our vehicle and sends distance information to the ACC controller. The ACC controller maintains a constant distance between the host and target vehicles. This is done using control of the throttle and breaks.

An important point is that the vision system must see the road and use lane markings to determine if targets in front are really in front - that is, if they are in our path. Then, it should "lock" on to the "primary" target, or as called in the industry the CIPV (Closest In-Path moving Vehicle). According to Mobileye's description of their system, the sensor's look-ahead characteristic allows estimation of the curvatures ahead of us, and this helps to have a stable lock on the target when the road is twisting. The system is claimed to perform well in heavy traffic, to be able to distinguish between moving and stationary targets, and to provide distance control for both types of targets.

Mobileye's system can work with a wide field camera. A wide field of view allows early detection of vehicles when they "cut-in". Cutting-in vehicles (or ones which seem to be cutting in) are detected at an early stage (as early as the car appears) and a cut-in signal is generated. This is done, as in the application described in this thesis, using motion detection and segmentation.

Target Tracking range	0-110 meters	0-220 meters
New target detection range	5-60 meters	5-110 meters
System operation frequency	30 fps	30 fps

TAB. 2.1 – MobiEye's ACC specifications.

FIG. 2.14 – Automatic vehicle guidance presented by the Center of Robotics of École des Mines de Paris, in IV2002 using vision. In the second image, we can see the driver's hands outside of the car, while the car is driving at about 50 km/h on curve.

In case of heavy rain or dense fog, Mobileye's system's can detect that it cannot work reliably. It then notifies the driver and turns itself off.

Table 2.1 provides some technical details of the Mobileye ACC system. One can clearly see that for detecting a new target the system needs a closer distance than to maintain one. This is a common feature of all detection systems.

2.3.7 Non-cooperative lateral control

Lateral control of vehicles (that is, automatic control of the wheel) is a delicate subject ; if such a system fails, someone might get killed... therefore the only technologically sound systems are those who cooperate with the infrastructure. Such cooperative systems exist for airplane (automatic guidance and landing) using ground beacons. For cars there are systems like the ones described in the next section.

In the term "Non-cooperative lateral control" we refer to systems who control the wheel of the car without having the need to installed something on the ground. Such a system was introduced by the Center of Robotics of École des Mines de Paris in the IV 2002 symposium in Versailles, France.

The system works on vision only using a frontal camera. It follows the two white lines of the road, and can cope well even if one line is absent. In the demonstration (see figure 2.14), the car drove over 100 km/h in the straight road and 50 km/h in curves. The vision system was also capable of directing a lane change (although at low speed)

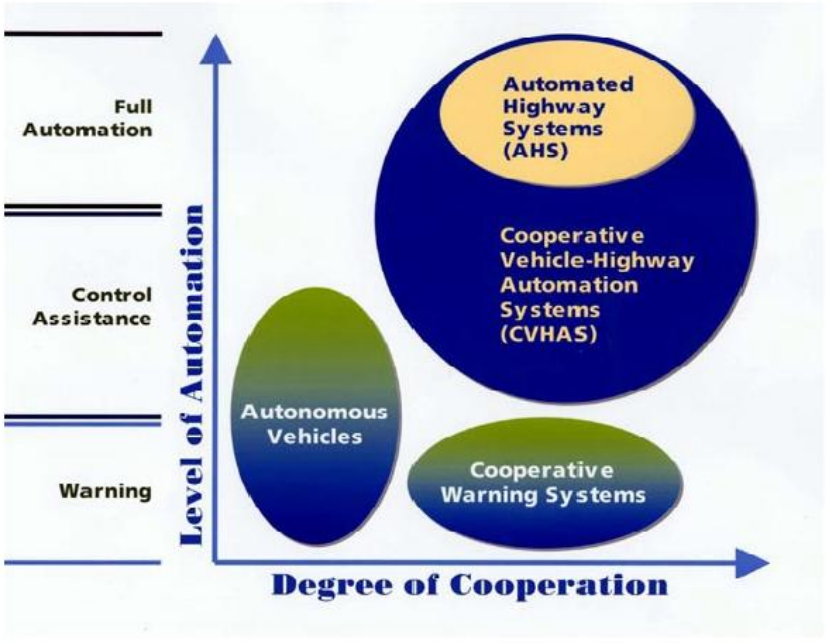


FIG. 2.15 – Different levels of vehicle-infrastructure cooperation

2.4 Vehicle-Infrastructure cooperative systems

Cooperative vehicle-highway systems offer the potential to enhance the effectiveness of independent vehicle applications as the ones described in section 2.3. These autonomous vehicle-based safety systems, even though they are effective, have some limitations based on two simple facts :

- Since they operate in an environment which is not supportive, they must employ non-cooperative detection modes such as camera and radar. Imagine an automatic landing of an airplane without the use of beacons in the airport, just by radar detection of the runway.
- Even when they "see" reliably the surroundings, they can't see around blind curves, for instance.

At the same time, autonomous infrastructure systems as the ones described in section 2.2 can detect dangerous situations in real-time and know about the structure of the road, but such infrastructure systems suffer from the limitation that they can only influence drivers who choose to pay attention to them. And even then, it depends on the drivers to make correct and fast decisions.

Cooperative intelligent vehicle-highway systems (CIVHS) offer an improved level of overall functionality by bridging this gap. These systems are cooperative in that the vehicles can receive information from the infrastructure and react accordingly, and vehicles can detect and report dangerous situations to the infrastructure, to be used by other vehicles. In a higher-level, vehicles can receive information about the geometry of the road and drive automatically, as in the AHS (Automatic Highway System) program. This program, whose story is brought

FIG. 2.16 – The AHS demonstration in San-Diego in 1997

in the next subsections, started in the United States in the early 1990s, and had the ambitious goal of fully automatic driving on passenger cars.

2.4.1 The AHS project

The idea of Automated Highway Systems (AHS) was very popular during the 1990s. The U.S. department of transportation (USDOT) sponsored a program which was done by the National Automated Highway System Consortium (NAHSC). The result of this work was shown in the Demo '97 in San Diego. In the demo, a group of vehicles drove automatically (i.e. without driver) on a highway, and arrived to high speed. This, in front of a large number of viewers, making a lot of impact on the media (see image 2.16).

However, USDOT canceled the NAHSC program in 1998. The reasons were lack of budget and the change of focus to more realistic ITS systems which would yield results in the near term. In the next subsection we will try to explain "whatever happened to AHS", as the name of the article by Richard Bishop [Bish]. But first, we will briefly review some technical aspects of AHS, according to official publications of the PATH project [Cali].

Longitudinal Control

In the platoon demonstration in San Diego in 1997, eight vehicles drove with close inter-vehicle distance with fully automated longitudinal and lateral control. The cars maintained a fixed spacing of 6.5 meters (21 feet) between them at all speeds up to full highway speed. The spacing was maintained with an accuracy of +/- 10 cm. (4 inches) during cruising and +/- 20 cm. (8 inches) during maneuvers like acceleration and deceleration. It is claimed that in the future improved spacing accuracy should reduce the spacing to less than 2 meters (6.5 feet).

This short spacing between vehicles can increase the throughput of the highway - the highway will carry three times as many vehicles (per lane per hour) than now. The other major advantages of the "platooning" system are better safety and less fuel consumption. Safety is becoming better by the automation of the vehicles (no place for driver human errors). Also, the fact that the relative speed between the cars (the difference between the speed of two adjacent cars) is very small inside the platoon. Because of this, even sudden high accelerations/decelerations cannot cause a serious damage.

Accurate spacing between the cars in the platoon is achieved by the longitudinal control system. This system uses radar and radio communication between cars : each car in the

FIG. 2.17 – Fully automated LeSabre on test track : lateral control

platoon uses its radar to measure the distance to the preceding car. The radio communication system provides each car with broadcast information on the velocity and acceleration of the preceding car and the lead car of the platoon. All of these signals are used by the longitudinal feedback control system to continuously decide what is the desired acceleration of the car. The throttle or the brake is then used to provide the desired acceleration. Apparently the system has some knowledge of the dynamic behavior of the throttle and brake actuators, so it can control it in a way that the desired acceleration is achieved accurately. The longitudinal control system updates the actuators at a rate of 50 times per second.

The platoon demonstration also showed how a car can leave or join the platoon. The radio communication system is used to coordinate such maneuvers within the platoon. A car that wants to leave the platoon informs the leading car. The leading car gives a permission for the process, and the car who wants to leave opens a larger space with the car in front and back. Then, the car makes a lane-change, and once it has exited the platoon, the other cars close the gap which was created. After that, the spacing between the cars in the platoon is preserved.

In the San Diego demo, the car that exited from the platoon returned to it after about one mile. The car exited from the front part of the platoon, slowed down and adjusted its speed to match exactly the tail of the platoon. Then it changed back to the lane of the platoon, and accelerated a little bit to stick to the platoon as the last car. Cars that enter the highway will always join platoons in this way.

Lateral Control

The AHS systems demands that the road will be equipped with some indicators that define its boundaries (see figure 2.17). The vehicle then uses a special kind of sensors to detect the indicators and to determine its location with respect to the road. The computer in the car uses this information to control the steering of the car and thus guide the car to follow the indicators.

The PATH automatic steering control system uses magnetic markers which are buried inside the road with distance of 4 feet between two consecutive markers. Laterally, they are positioned in road center. The polarities of the magnets can be changed and thus send information to the car. Information that is transmitted in this way is road geometry information, entrance and exit information, etc.

Six three-axis fluxgate magnetometers, developed by the company "Applied Physics",

located below the front and rear bumpers of the host vehicle, detect the magnetic field of the magnets. A signal processing algorithm compares the magnetic strength to the virtual 'magnetic field map' of the magnet, that it has, and eliminates the background noises. Then it determines the relative position of the vehicle to the road center. This information is passed to a PC computer that runs the algorithm that decides which steering angle we need. The desired steering angle is sent to a servo motor which was added to the steering system (the motor was developed by the company "Delphi Saginaw"). This way the car is driven according to the indicators on the road.

The resulting system tracks the indication on the road with a maximal error of about 8 cm, and the passengers enjoy good steering comfort (no sharp movements). The system can be economical, because the price of the magnets is not high (per km). It is of course more expensive than a white line, but the results are much more robust. The system works well under all weather conditions such as rain, snow, and low visibility, since it is not using vision.

Fault Management

A very important feature of the vehicle control system is the automated fault management system that was implemented on the cars. The fault management system should detect and handle failures in the sensors of the car or in its actuators.

Usually, the failure is reported to be detected within a small time (0.1 seconds). If the failure is simple, the driver is even not informed. Otherwise, for example when the radio communication is not working, then the driver is informed and the space between cars is being widen to 15 meters. If an actuator stopped working, the driver is informed and the automatic control shuts off.

When one car has a failure, all the other cars are informed. Even if the car's computer crashes, the other cars will know it, probably because they are using a "heart beat" signal (not fully clear from the PATH publications).

2.4.2 The deployment problem

Whatever happened to AHS

AHS is a great test case of the problems that prevent vehicle-highway cooperative ITS application from massively diffusing into the industry. The AHS system worked properly; it did not introduce any unsolvable problem. However, the USDOT support was stopped in 1998 with the feeling that one could not afford to block a whole lane in the highway, when the overwhelming majority of the cars are not equipped with the appropriate devices. This is the deployment problem, in its simplicity : the cars industry will not equip cars with expensive devices before infrastructure is ready; the highway managers could not justify an expense of preparing the infrastructure before a large amount of cars are equipped with appropriate devices. In the case of AHS, "preparing the infrastructure" means adding a dedicated lane (or using an existing lane, which is equally unaccepted) - definitely a large expense.

The future of vehicle-highway cooperative systems

According to Richard Bishop [Bish], the next thing are cars which are operating in today's roads with no complicated cooperation from the infrastructure or no cooperation at all. This is what he calls the first generation. Then, he estimates that co-pilots (driver assistance) will develop to full pilot (full driving automation). We agree with Bishop. We think that after the penetration of non-cooperative IV applications into vehicles, simple cooperation between cars and infrastructure will start to emerge.

For instance : a vision-based system will guide vehicles fully automatically, but only on specific roads where the white line will be redrawn, and some radio communication will mark the entrance to the "automatic zone". The road will contain mixed traffic (also regular cars) but as time goes on, more and more cars will be automatic in such roads, and more and more roads will have this equipment. The system that these cars will have will be a developed version of LDW.

The distance between cars in such "semi-automated roads" will be, also according to Bishop, smaller and smaller.

Even though non-automated cars will be able to travel around for many years from now, they will be restricted. Automated cars could enter high-occupancy vehicle (HOV) lanes, and this will create motivation to have such a car. The results will be a small improvement in capacity of roads, a certain improvement in safety and in traffic flow.

Once these systems are mature and widely used, we (or our grandchildren...) will start to see more and more dedicated roads which are forbidden for "manual" cars. More and more items in the infrastructure will be designed for automated vehicles. All this will happen, to our opinion, only in long segments of roads (between two junctions) and never in a road that contains a junction. The driving in a junction will remain manual, to our opinion, much longer than on restricted road segments. The form will be "leave the junction manually, and get into auto-pilot. Once a junction arrives, the auto-pilot shuts off".

Regarding the time scale, we agree with the estimation of Bishop :

- Until 2010 : wide use of non-cooperative IV applications like the ones described in section 2.3.
- Until 2025 : wide use of on-boards systems which are simply cooperating with the infrastructure - mainly on restricted road segments. Very beginning of restriction on non-automated vehicles (fully-dedicated lanes).
- Far future (perhaps 2050) : Fully dedicated lanes in broad use, applications working to some extent also in urban areas and junctions.

Bishop indicates that according to the last intelligent vehicles congresses, it is clearly demonstrated that driver assistance systems have graduated from the R&D phase to real product development. He says that both European and Japanese car-makers have active programs focused on automated driving. As described above, these projects will fit well with the safety and convenience systems which have arrived or will arrive to the market, like ACC and LDW.

2.5 Choice of applications

This thesis deals with application using a single frontal on-board camera. There are many applications that use such equipment. In this section, we explain the choice of the two applications we developed.

2.5.1 List of possible applications

Let us list some applications that were mentioned in this chapter. All these applications are based on the use of one camera associated to an image processing hardware.

Obstacle detection

A camera is fixed behind the window looking forward the vehicle. The goal is to detect any obstacle located a few meters in front of the vehicle [1]. This function is used by the 3 applications described below :

- **Low speed obstacle detection** : A camera is fixed behind the window looking forward the vehicle. The goal is to detect any vehicle located a few meters in front of the vehicle in urban conditions. Then the system will calculate the distance between the vehicle and the car detected. This application may help the driver in certain traffic conditions in order to decrease his stress and his tiredness [1]. It may also help him to deal with the complexity of the situations including cut-in scenarios.
- **Pedestrian detection** : A camera is fixed behind the window looking forward the vehicle. The system must detect pedestrians crossing the street few meters in front of the vehicle in order to have enough time to activate any electronic system such as "pedestrian airbag" [Bert 00][Gavr 01]. The goal is to reduce the injuries of the pedestrians bumped by a car. It is an application of obstacle detection using classification.
- **Stop&Go ACC** : A camera is fixed behind the window looking forward the vehicle. The goal is to detect any vehicle located a few meters in front of the vehicle in urban conditions. Then the system will calculate the distance between the vehicle and the car detected. This distance is used to regulate the car's speed in order to keep a steady headway to the car in front.

Lane detection

A camera is fixed behind the window looking forward the vehicle. The system has to detect the white lines marked on the road. Then it calculates the distance between the vehicle and the lines [3]. The purpose is to warn the driver to help him to avoid a run-off-road. The warning interface may be acoustic or tactile.

Driver recognition

A camera is looking at the driver to recognize him in order to do the seating adjustment and / or the mirrors adjustment and / or the steering wheel adjustment automatically.

Fall asleep warning

A camera is looking at the eyes of the driver. The system warns the driver if the eye blinking means that the driver falls asleep. The goal is to reduce accidents due to drowsiness and tiredness. The difficulty of this function comes from the fact that it has to detect few seconds before the driver falls asleep. Another question is how to wake the driver up.

Blind spot detection

A camera is fixed in the side mirror looking backward. The goal is to detect vehicles in the blind spot which are not visible in the side mirror [4]. The purpose is to increase the visibility of the driver and to reduce lateral collision during lane change maneuver.

Curve detection

A camera is fixed behind the window looking forward the vehicle. The system will measure the orientation of the road especially the road curvature [5]. Regarding other typical systems used to measure the curvature such as gyroscope, the advantage of this system corresponds to its capability to predict the type of the curve. The aim of the application is to reduce accidents due to overspeed.

Climatic environment measurement

A camera is fixed behind the window looking outside the vehicle. The goal is to measure different climatic conditions like rain or brightness to activate automatically the wipers and the lights.

2.5.2 Applications quotation

The purpose is to evaluate the different functions listed in the previous paragraph in term of customer interest, compatibility with the pre-existing algorithms available in a standard video encoder such as motion estimation, feasibility and benefits in order to choose the two functions developed in this thesis.

Description of criteria

- **Customer interest** : This criterion corresponds to the will of a customer to buy this application and the possibility for a car manufacturer to earn money regarding the price the customer is ready to pay and the cost of such a function. The values are shown in table 2.2.
- **Compatibility** : This criterion is the level of compatibility of the algorithm needed for the function with the pre-existing algorithms available in a standard video encoder such as motion estimation. The value goes from 1 (bad) to 5 (very good).
- **Feasibility** : This criterion takes into account the possibility to develop this algorithm and to integrate it into a specific hardware. The value goes from 1 (bad) to 5 (very good)

Value	Description
3	The customer agrees to pay more to get the function and the price allows the firm to earn money
2	The customer agrees to pay more to get the function but the price does not allow the firm to earn money the first years
0	The function will be enjoyed by the customer but he does not agree to pay more to have it
-3	The function will not help the customer

TAB. 2.2 – Description of values.

Function	Customer interest	Compatibility	Feasibility	Benefits	TOTAL
Low speed obstacle detection	3	5	5	5	18
Pedestrian detection	2	5	5	5	17
Lane detection	0	3	4	1	8
Driver recognition	0	2	2	2	6
Fall asleep warning	0	2	2	2	6
Blind spot detection	0	3	3	1	7
Curve detection	2	2	3	4	11
Climatic environment measurement	2	1	2	3	8

TAB. 2.3 – Quotation.

– **Benefits** : The benefits mean the interest for car manufacturers to develop this function facing their regulations and strategic policy. The value goes from 1 (bad) to 5 (very good)

Quotation

The applications developed in this thesis were developed in cooperation with (among others) Renault research division. The quotation of the different applications is based on Renault experience of driving assistance systems and on studies of accident statistics and situations. It also takes into account the new regulations in the next years. The quotation appears in table 2.3.

In this thesis we chose to implement the two applications that have the higher score, namely pedestrian detection and low speed obstacle detection. For low speed obstacle detection,

we choose to detect vehicles and use it for a low-speed ACC application.

2.6 Choice of implementation methods

2.6.1 Requirements for general detection and tracking applications

Both applications are basically detection and tracking systems. Coming to design such a system, we should analyze the input and output of the system. The main input of the system - the images arriving from the camera - can be analyzed by several observations. Each one of these observations detects different features of the image - motion, dark areas, symmetries, vertical edges etc. The output, on the other hand, should be a list of targets, with accurate spatial position of each target, plus a measure of this accuracy (=confidence value). The question, therefore, is how to go from the input observations to the output.

The problem is difficult :

- Image processing is a non-linear and non-gaussian observation process.
- Projection in an image is also non-linear and non-gaussian.
- Simple Kalman filter (typical algorithm for model inversion) is out of focus here. Extended Kalman filter might operate poorly (it is not known what model to provide).

2.6.2 Previous approaches

Bayesian Networks

Bayesian Networks-based data fusion was used in a previous project (FADE 1, see [Steu 02]), and provided good results. Unfortunately, this method exhibits severe limitations, as discussed below.

Advantages :

- This is a diagnosis approach : enables the detection of failing algorithms (especially image processing algorithms), through the computation of correlations between proposals made up from several sources.
- Observation is completely independent of the ground plane constraint, since we use a model of the width of the target for retro-projection (typically 1.75m).
- Accuracy of localization is high, since we combine many different "sensor outputs" from many different algorithms, thus yielding in a typical 10% distance accuracy.

Inconveniences :

- One target = one hypothesis (one state space position). No real state space exploration.
- Bayesian nets don't allow to distinguish between confidence and accuracy of localization of the target. When a hypothesis gets a low score, it is difficult to state whether the target is badly located (inaccuracy of detection) or the target simply doesn't exist or doesn't fit the model (uncertainty). It's very difficult to set up a "split" strategy from the results provided by the Bayesian net.
- The major disadvantage is that the actual setup can only use image processing algorithms that provide vertical features (in order to combine inputs). This is good for edge

detection, lights, shadow, symmetry detection but how to integrate motion segmentation into that framework ?
- The fusion algorithm is difficult to tune. We have to attribute a priori belief to each source, which was found to be tricky.
- Bayesian net computation is heavy, so it's generally necessary to get back to approximations of Bayesian nets (which were done in FADE 1), which to some extent tends to deteriorate the results.

Problems :
- It is unclear how to integrate motion segmentation into such a framework.
- It is not possible to cope with different kinds of targets, like motorcycles and trucks.

2.6.3 Particle filtering

Particle filtering is a useful method to track targets in non-linear, non-gaussian environment [Douc 02], as is our system. Particle filtering is a technique for implementing a recursive Bayesian filter by Monte Carlo simulations. The reader is referred to chapter 5 for a complete overview.

Compared to the Bayesian nets approach, using particle filtering has several advantages :
- One target = n hypotheses. This is the basis for the algorithm, and it makes fusion and splitting between targets more natural. Accuracy of localization should be high, due to the high number of particles. Averaging these positions (according to their weight) yields the global target position output.
- Visual results : eases the tuning. Tuning is done through likelihood functions, which can be of any sort (including non-linear functions) and thus are very adapted to image processing algorithms.
- Allows to distinguish between confidence (weights of particles) and inaccuracy (spatial distribution of particles).
- Allows the integration of many different algorithms (of any kind, providing any kind of input). Makes possible the integration of horizontal edge detection and the use of different models than just the width of the car (i.e. height of lights at short distance, by using the hypothesis of plane road, which is accurate at very low distance).

2.6.4 The need for motion estimation

When detecting objects in scenes, it is essential to use the motion information ; that is, the information that is discovered by comparing several images.

To understand why motion is needed, look at figure 2.18. This image, taken from an on board frontal camera, contains something that was classified by a detector as "pedestrian". From a first look at the static image, it is not clear whether this detection is correct. Only while looking at the moving video (which cannot be included in this thesis for understandable reasons, see figure 2.19 for one image out of this video) one can see that this is a true pedestrian, detected only by his upper part. He is colliding at that point of time with the right part of a car and with a street light, and in fact, only the head belongs to him. The "body" (seen in figure 2.18) is the car and the street light.

FIG. 2.18 – A pedestrian or not ?

FIG. 2.19 – The wider image

2.6.5 The need for a learning algorithm

Cars detection can go quite well with traditional algorithms, such as rear lights and plate detection, symmetry, vertical edges etc. When dealing with pedestrian detection, however, there are not many algorithms which can perform well for this problem. For this purpose, we need a learning algorithm that will adapt itself to the variable shapes of pedestrians.

Among learning algorithms, we preferred AdaBoost over SVM and neural networks (two popular learning algorithms), because of the following reasons :

1 AdaBoost is easy to implement and understand.

2 For AdaBoost, there is proof of convergence that shows exactly what measured results one can obtain.

3 AdaBoost can be implemented in the cascade method of Viola and Jones (see details in chapter 3) and works faster than any other algorithm.

4 There are possibilities to combine AdaBoost easily with hardware, thanks to the freedom to choose whatever we want as "weak classifiers".

More information about learning algorithms, AdaBoost and how we used it is given in chapter 3.

2.7 Conclusion

In this chapter, we have presented a comprehensive review of the current situation in the world in the domain of ITS. We have seen that efforts are ongoing in many countries, both in car manufacturers and in public authorities. In particular we have presented the work being carried out in our laboratory.

Based on this review, we described how we chose the ITS applications of this thesis. We explain the motivation of building these applications as well as of choosing the specific implementation methods.

In the next chapters we will describe the algorithms that allow us to implement these applications.

Chapitre 3

Visual object detection with machine learning

When we speak about object detection, we mean that we take an image and would like to detect if and where there is an object of a certain type. In our case relevant objects are pedestrians or cars[1].

In this chapter, we discuss systems that learn object detection. Such a system is using a large amount of negative and positive examples in order to learn the shape of a specific object. Such examples are simply small image files that contain the object (for positive) or without the object (for negative).

We will begin this chapter by several conventions and definitions, followed by the state of the art existing in this domain. Then, we will present our contribution to this field.

3.1 Conventions and definitions

3.1.1 Motion information

In this work, unless stated otherwise, we speak about only static detectors. This means, that the detector works on each image separately and finds the locations of the objects, without relation to previous images.

3.1.2 Detection window

Each detector is associated with some conventions, which should be determined prior to the learning process, and even prior to the collection of the learning data. The conventions define the way in which the pedestrian is related to the detection sub-window. In our implementation, we used for pedestrians a 1 :2 rectangle positioned on the pedestrian with about 10% margins from each side (see Figure 3.1). For cars, we used a 1 :1 rectangle with 15% margins (see 3.2).

[1]Much of the work presented in the chapter is relevant as well for other types of objects, such as human faces.

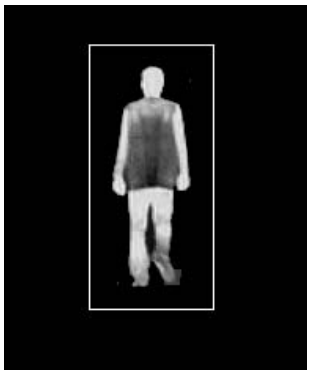


FIG. 3.1 – The pedestrian conventions : the detection window's height is twice as large as its width, and the pedestrian leaves margins of about 10%.

FIG. 3.2 – The car conventions : the detection window's height is the same size as its width, and the car leaves margins of about 15%.

3.1.3 Measuring the detection rate of a detector

Ground truth evaluation

To measure the success of a detector on a given image, we need *ground truth* information. This means, that all the pedestrians in an image will be exactly marked by hand prior to the evaluation of the detector. The ground truth indicates *where the pedestrians really are*. The ground truth for image I is given as a (possibly empty) list of rectangles $g_1, g_2 \dots g_m$, positioned on the pedestrians in the image, according to the same conventions mentioned in the previous section.

To evaluate the detector, we apply it on the image. The detector returns a (possibly empty) list of detected rectangles $d_1, d_2 \dots d_n$. We now have to compare these rectangles to the ground truth to measure the success. For this, we define the following simple definitions :

1 Two rectangles r_1 and r_2 are said to be *close* if :
- $\mid L(r_1) - L(r_2) \mid < 0.3 * max(W(r_1), W(r_2))$
- $\mid R(r_1) - R(r_2) \mid < 0.3 * max(W(r_1), W(r_2))$
- $\mid T(r_1) - T(r_2) \mid < 0.3 * max(H(r_1), H(r_2))$
- $\mid B(r_1) - B(r_2) \mid < 0.3 * max(H(r_1), H(r_2))$
- $\frac{2}{3} \leq \frac{W(r_1)}{W(r_2)} \leq \frac{3}{2}$

- $\frac{2}{3} \leq \frac{H(r_1)}{H(r_2)} \leq \frac{3}{2}$
  where $L(r), R(r), T(r), B(r), W(r), H(r)$ are the left, right, top, bottom, width and height of a rectangle r, respectively.

2 A ground truth location g is considered as *detected* if there exists a detection $d \in d_1, d_2 \ldots d_n$ for which g and d are close.

3 A detection d is a *false detection* if there is **not** any ground truth location g for which g and d are close.

Definition 1 above is, of course, arbitrary. One has to choose reasonable values in order to decide when two rectangles are close. In definition 1 we verify that the 4 borders are close (up to 30% of the width or height) and that the dimensions are similar (up to 50%).

For a given image, using these definitions, we can determine F, the number of *false detections* among $d_1, d_2 \ldots d_n$, and S, the relative part of the ground truth which are *detected*.

In the literature, the false detection rate is sometimes expressed as a percentage of the total number of examined sub-windows. For simplicity, we deal here with absolute numbers.

These two numbers - the detection rate and the false detection rate - are the only information needed to evaluate a detector in terms of detection success.

The ROC curve

The two numbers mentioned in the previous section must come together. No one of them alone is sufficient. For examples : if I told you I have a detector that detects 100% of the pedestrians in an image, is that good ? not necessarily ; it can be a detector that "detects" in each image all the possible sub-windows, therefore having many false detections. In order to really evaluate a detector we must say, for example that "it detects 85% of the pedestrians while having 3 false detections per a 640x480 image".

The sentence above, however, reveals only a part of the characteristics of the detector. We want to know also how many false detections are obtained if we tune the detector to detect 90% and 95% of the pedestrians, etc. The tool to explore the full space of information is the Receiver Operating Characteristic (ROC) analysis. In this analysis we draw a curve for all the matching values of detection rate and false positive rate.

3.2 State of the art

3.2.1 Support vector machine (SVM)

SVM is a technique which is known for many years [Vapn 95] as a useful mechanism for machine learning. Given an object class, SVM is capable of learning this object class through a set of positive and negative examples. If, for examples, the object class is a pedestrian image, then the SVM can learn this object class provided that it has access to a large set of positive (=pedestrian images) and negative (images not containing pedestrians) examples. This is the *training process*.

The result of the training process is a *binary classifier*. Given a new object, the classifier can tell if it belongs to the object class or not. In our example : given a image, it can tell,

up to some error, if it contains a pedestrian or not. The error, of course, depends on many factors, like the intra-class variability (how objects inside the class differ from each other), the amount of training examples, the parameters of the SVM and more.

We will now see a brief explanation of SVM. The following is based on [Burg 98].

In this introduction we limit ourselves to linear support vector machine.

SVM is based on the idea of seeing the data as points in a multidimensional space and separating two sets of points (i.e. negative and positive examples) by a hyperplane . Suppose we have N training data points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ where $x_i \in \Re^d$ (every type of data can be translated into some d dimensional space) and $y_i \in \{1, -1\}$ (1 for positive examples, -1 for negative). We would like to learn a linear separating hyperplane classifier :

$$f(x) = sgn(w \cdot x - b) \tag{3.1}$$

Meaning that the hyperplane having normal w and factor b is separating the positive and negative points. Furthermore, we want this hyperplane to have the maximum separating width with respect to the positive and negative data points. That is, we want to find a hyperplane $H : y = w \cdot x - b = 0$ and two "side" hyperplanes parallel to it and with equal distances to it,

$$H_1 : y = w \cdot x - b = +1 \tag{3.2}$$

$$H_2 : y = w \cdot x - b = -1 \tag{3.3}$$

with the condition that there are no data points between H_1 and H_2, and the distance between H_1 and H_2 is maximized.

Note that the values of $+1$ and -1 are always possible - that is, for any separating plane H and corresponding H_1 and H_2, we can always "normalize" the coefficients vector w so that H_1 will be $y = w \cdot x - b = +1$, and H_2 will be $y = w \cdot x - b = -1$.

We want to maximize the distance between H_1 and H_2, so there will be some positive examples *exactly on* H_1 and some negative examples exactly on H_2. These examples are called support vectors because only these points actually participate in the definition of the separating hyperplane, and other data points can be removed or changed as long as they don't cross the planes H_1 and H_2.

Recall that in the two dimensional case the distance from a point (x_0, y_0) to a line $Ax + By + C = 0$ is $\frac{|Ax_0+By_0+C|}{\sqrt{A^2+B^2}}$. On the d dimensional case, the distance of a point on H_1 to $H : w \cdot x - b = 0$ is $\frac{|w \cdot x - b|}{\|w\|}$, and the distance between H_1 and H_2 is $\frac{2}{\|w\|}$. So, in order to maximize the distance, we should minimize $\|w\| = w^T w$ with the condition that there are no data points between H_1 and H_2 :

$w \cdot x - b \geq +1$, for positive examples $y_i = +1$

$w \cdot x - b \geq -1$, for negative examples $y_i = -1$

These two conditions can be written together as one condition :

$$y_i(w \cdot x_i - b) \geq 1 \tag{3.4}$$

So our problem can be expressed by the following formula, which is a convex quadratic programming problem :

$\min_{w,b} \frac{1}{2} w^T w$, subject to $y_i(w \cdot xi - b) \geq 1$

We can use Lagrange multipliers $\alpha_1, \alpha_2, \ldots, \alpha_N \geq 0$, and then we have the following Lagrangian :

$$\mathcal{L}(w, b, \alpha) \equiv \frac{1}{2} w^T w - \sum_{i=1}^{N} \alpha_i y_i (w \cdot x_i - b) + \sum_{i=1}^{N} \alpha_i \tag{3.5}$$

To proceed, we can solve the Wolfe problem which is equivalent to our problem : maximize $\mathcal{L}(w, b, \alpha)$ with respect to α, subject to the constraints that the gradient of $\mathcal{L}(w, b, \alpha)$ with respect to the primal variables w and b will disappear :

$$\frac{d\mathcal{L}}{dw} = 0 \tag{3.6}$$

$$\frac{d\mathcal{L}}{db} = 0 \tag{3.7}$$

and that :

$$\alpha \geq 0 \tag{3.8}$$

From Equations 3.6 and 3.7, we have

$$w = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{3.9}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{3.10}$$

Substitute them into $\mathcal{L}(w, b, \alpha)$, we have

$$\mathcal{L}_D \equiv \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \tag{3.11}$$

in which the primal variables are eliminated. When we solve α_i, we can get $w = \sum_{i=1}^{N} \alpha_i y_i x_i$, (we will later show how to compute the threshold b). Now, when we have a new object x to classify, all we have to compute is :

$$f(x) = sgn(w \cdot x + b) = sgn((\sum_{i=1}^{N} \alpha_i y_i x_i) \cdot x + b) = sgn(\sum_{i=1}^{N} \alpha_i y_i (x_i \cdot x) + b) \tag{3.12}$$

It is important to observe that in this calculation, all the training data points x_i appear only in the form of dot product. This implies that for classifying an object it is enough to know a function (i.e. the dot product with the data points) and not the data points themselves. This can save computations.

FIG. 3.3 – The top row shows examples of images of pedestrians in the training database of Papageorgiou et al. The examples vary greatly in color, texture, and background. The bottom row shows the corresponding edge maps generated by a Sobel filter

3.2.2 Wavelets features for visual object detection

In this subsection we review the work of Papageorgiou et al [Oren 97] [Papa 98a] [Evgc 00] [Papa 99] [Moha 01], which tried to solve the problem of pedestrian detection using SVM. These authors have developed a general, trainable object detection system that can find pedestrians in rather complex images. Their system learns from examples, each example is an 128x64 pixels image of a pedestrian or non-pedestrian.

The work of Papageorgiou et al is unique and was chosen to be mentioned here because most previous systems designed to detect pedestrians in video sequences have used hand-crafted models and/or motion information. Hand-crafted models limit systems to be used only for the specific case (in this case, pedestrians) and can not be ported to other cases. The assumption that all pedestrians are moving is of course not correct in the general case and is very problematic when camera is moving, which is the case in the context of this thesis. Discussions of some of the systems mentioned above can be found in [Tsuk 85][Leun 87][IHar 98][Bert 02].

The system of Papageorgiou et al is based on a novel object representation that uses projections of the object images onto a space with less dimensions - what they call *a dense Haar wavelet basis*. This projection efficiently encodes the structure of the pedestrians, in two different scales. The set of coefficients they use contains 1326 components - and this is obviously much less than the dimension of an 128x64 image, which is 8192. These features are used to train a SVM classifier. With this representation, their system can reliably detect pedestrians in static images with no motion information. In further sections we will see how motion information can be combined.

Transforming images to wavelet representation

One of the important things in object detection is what object representation to use. We search for the representation that yields high inter-class variability (between pedestrians and themselves) and low intra-class variability (between pedestrians and non-pedestrians). Figure 3.3 shows several example images of pedestrians from the training set used by these authors. From these images, we can see that a pixel-based representation would give poor

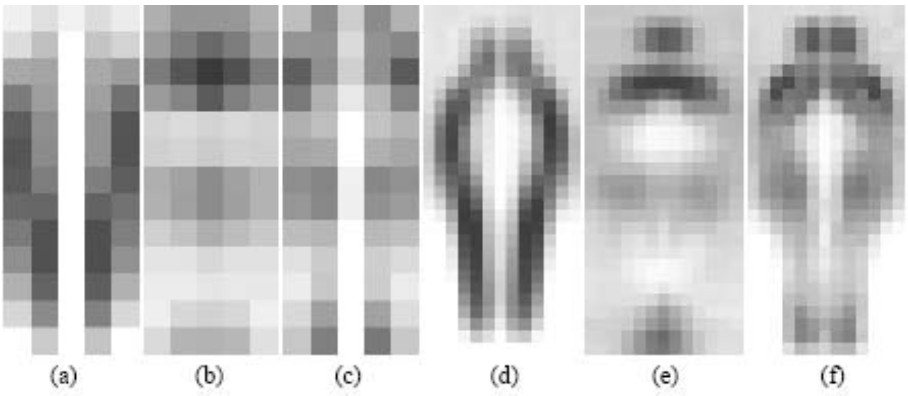


FIG. 3.4 – Ensemble average values of the wavelet coefficients coded using gray level. Coefficients whose values are above the average are darker, those below the average are lighter. (a)-(c) vertical, horizontal, and diagonal coefficients at scale 32

results because of the high variability in the color of the pedestrian patterns. As seen in the lower line of the image, a traditional edges representation is also not so good. In the figure we can see how the processing to edges in the images have very little consistency between the patterns of the pedestrians and a lot of noisy information which has nothing to do with the shape of the pedestrian.

Actually all the approaches that use color would have the same problems because there is no consistent color information. The representation used by Papageorgiou et al tries to overcome these problems by looking at *intensity differences* in small local regions. These differences are called Haar wavelets [Mall 89]. The Haar wavelet transform goes over an image and calculates a set of coefficients at different scales - each coefficient measures the response of the wavelet feature by calculating the difference between two rectangular regions. Wavelets appear in three types : vertical, horizontal, and diagonally oriented. So the transform being performed on the image yields three sets of coefficients, one for each of the wavelet orientations. Since these authors used color images, they run the Haar transform over each color channel separately. Then, for a given spatial location and orientation, they choose as the wavelet coefficient the one from the three color channels that gives the maximal response. To reduce computation by a factor of 3, they can use only intensity images. Of course, as seen towards the end of this section, results of the detection will be less successful in this case.

Manual reduction of wavelets

The training database used by Papageorgiou et al is a set of 1848 frontal and rear color views of pedestrians (924 plus mirror images) that have been scaled to 128x64 pixels and aligned such that the pedestrian is in the center of the image. The data is available on the web and indeed in further chapters we will use this data set for our experiments. Using the Haar wavelet representation, they look at both coarse-scale (32x32 pixels) and fine-scale (16x16 pixels) features. At these scales of wavelets, there are 1326 total features for a 128x64 pattern. Many of these coefficients will be irrelevant for the task of pedestrian detection (e.g.

the coefficients at the corner areas that do not touch the pedestrian's body), therefore in some stage of their work they obtained a much smaller representation (29 instead of 1326) by choosing a small subset of the coefficients that are consistently strong or weak.

Average responses for each coefficient for the three orientations and the two scales are shown in Figure 3.4. In the figure, a dark coefficient indicates consistently strong response and a light coefficient indicates consistent weak response. If we want to choose a subset of wavelets as mentioned above, we manually choose 29 of these coefficients. This way each pedestrian image is represented by a 29-dimensional feature vector.

SVM usage

the SVM classifier is trained using the 1848 pedestrian patterns and a set of 7189 negative patterns collected from images of outdoor scenes not containing pedestrians. The advantage of SVM that these authors took advantage of, is that SVM, instead of minimizing the training error of a classifier (i.e. the error on the training set, which often doesn't say much about the results on other sets), minimizes an upper bound on its generalization error This is actually the distance between the two hyperplanes we saw in the previous section.

To detect pedestrians in a new image, Papageorgiou et al shift the 128x64 detection window over all locations in the image. Of course, this will only detect pedestrians at a single scale. To perform multi-scale detection, they incrementally resize the image (creating a so-called "pyramid") and run the detection window over each of these resized images[2].

Results

To evaluate the performance of a detection system, it is necessary to analyze the full ROC curve which gives an indication of the tradeoff between detection rate (i.e. how many of the pedestrians are successfully detected) and the number of false positives. Of course, the ROC curves are computed over an out-of-sample test set (gathered by the authors around MIT and over the Internet). Figure 3.5 compares the ROC curves of several different versions of the system. They are as follows :

- color processing with 29 features using a homogeneous polynomial of degree two.
- color processing with 29 features using a polynomial of degree two.
- color processing with 29 features using a polynomial of degree three.
- grey-level processing with 29 features using a homogeneous polynomial of degree two.
- grey-level processing with 29 features using a polynomial of degree two.
- grey-level processing with 29 features using a polynomial of degree three.
- color processing with all 1326 features using a polynomial of degree two.

[2]this method is common in all the work we will review hereinafter. In the case of one work (Viola and Jones, see further sections) the authors succeed to scale the feature and not the image. This saves them the time needed to create this "pyramid". However scaling a feature is not always convenient because the pixels locations are discrete (you cannot increase a distance of 4 pixels by 10%). One simple but tedious solution is to rerun the learning over all scales (each time the training set is scaled to another scale), and actually produce an independent detection system for each different scale. This solution seems to be perfect, but it is left for future research.

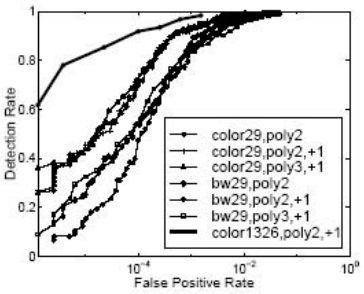


FIG. 3.5 – ROC curves for different detection systems. The detection rate is plotted against the false positive rate, measured on a logarithmic scale. The false detection rate is defined as the number of false detections per inspected window.

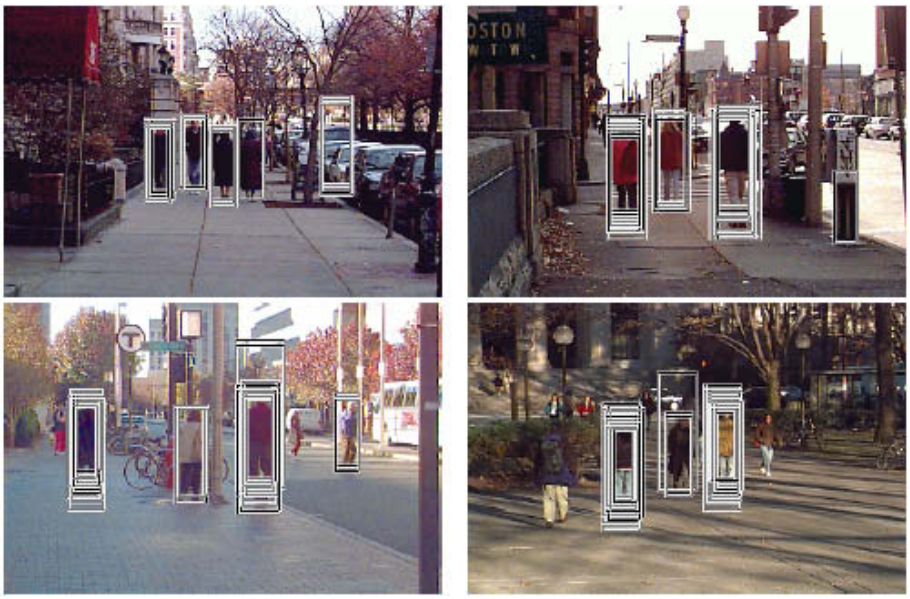


FIG. 3.6 – Results from the pedestrian detection system of Papageorgiou et al.

From the ROC curve, it is clear that the type of the features (color versus no color and 29 versus 1326 coefficients) makes most of the difference on the performance. The complexity of the classifier (homogeneous/non homogeneous, degree two/three) is secondary. As expected, using color features results in a more powerful system. The curve of the system with no feature selection is clearly better than all the others. This indicates that if one wants the maximum performance, using all the features is optimal.

It may be possible to achieve the same performance as the 1326 feature system with fewer features (but more than the 29 used here); this was an open question at the time of that work, and was solved using the AdaBoost algorithm, as described in further sections. In Figure 3.6 we see examples of some outdoor images that were processed by the system of Papageorgiou et al. These images were not part of the training set.

3.2.3 AdaBoost

To understand AdaBoost we first need to understand the notion of *boosting*. To do that, we might use the following nice example which is taken from [Freu 99].

A horse-racing gambler, hoping to maximize his winnings, decides to create a computer program that will accurately predict the winner of a horse race based on the usual information (number of races recently won by each horse, betting odds for each horse, etc.). To create such a program, he asks a highly successful expert gambler to explain his betting strategy. Not surprisingly, the expert is unable to articulate a grand set of rules for selecting a horse. On the other hand, when presented with the data for a specific set of races, the expert has no trouble coming up with a "rule of thumb" for that set of races (such as, "Bet on the horse that has recently won the most races" or "Bet on the horse with the most favored odds"). Although such a rule of thumb, by itself, is obviously very rough and inaccurate, it is not unreasonable to expect it to provide predictions that are at least a little bit better than random guessing. Furthermore, by repeatedly asking the expert's opinion on different collections of races, the gambler is able to extract many rules of thumb.

In order to use these rules of thumb to maximum advantage, there are two problems faced by the gambler : First, how should he choose the collections of races presented to the expert so as to extract rules of thumb from the expert that will be the most useful ? Second, once he has collected many rules of thumb, how can they be combined into a single, highly accurate prediction rule ?

Boosting refers to a general and provably effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules of thumb in a manner similar to that suggested above.

Background

The following summary is based on [Freu 99].

Originally, boosting started from a theory of machine learning called "Probably Approximately Correct" (PAC) [Vali 84] [Kear 94]. This model allows us to derive bounds on estimation error for a choice of model space and given training data. In [Kear 88] [Kear 89] the question was asked if a "weak" learning algorithm, which works just slightly better than random guessing (according to the PAC model) can be made stronger (or "boosted") to create a "strong" learning algorithm. The first researcher to develop an algorithm and to prove its correctness is Schapire [Scha 89]. Later, Freund [Freu 90] developed a more efficient boosting algorithm that was optimal but had several practical problems. Some experiments with these algorithms were done by Drucker, Schapire and Simard [Druc 93] on an OCR task.

The algorithm

The AdaBoost algorithm was introduced in 1995 by Freund and Schapire [Freu 95]. AdaBoost solved many of the practical problems that existed in the previous algorithms mentioned above. In figure 3.1 we bring the original AdaBoost algorithm. The algorithm takes as input a training set $\langle x_1, y_1 \ldots x_m, y_m \rangle$ where each x_i belongs to some *domain* X (as in the case of SVM - any kind of training examples can be translated to a d dimensional space,

for some d) and each label y_i is in some label set Y. To our purposes we need only binary classification and thus can assume $Y = \{+1, -1\}$ as in the case of SVM above. Generalization to the multi-class case is not discussed here.

AdaBoost calls a given *weak* learning algorithm repeatedly in a series of "cycles" $t = 1 \ldots T$. The most important idea of the algorithm is that it holds at all times a distribution over the training set (i.e. a weight for each example). We will denote the weight of this distribution on training example $\langle x_i, y_i \rangle$ on round t by $D_t(i)$. At the beginning, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set.

The role of the weak learner is to find a weak rule (classifier) $H_t : X \rightarrow Y$ appropriate for the distribution D_t. But what is "appropriate" ? The "quality" of a weak classifier is measured by its error, according to the weights :

$$\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i) \tag{3.13}$$

The error is measured, of course, with respect to the distribution D_t on which the weak learner was trained, so each time the weak learner finds a different weak classifier. In practice, the weak learner is an algorithm - doesn't matter which algorithm - that is allowed to use the weights D_t on the training examples.

Coming back to the horse-racing example given above, the instances x_i correspond to descriptions of horse races (such as which horses are running, what are the odds, the track records of each horse, etc.) and the label y_i gives a binary outcome (i.e., if your horse won or not) of each race. The weak classifiers are the rules of thumb provided by the expert gambler, where he chooses each time a different set of races, according to the distribution D_t. Here is the "AdaBoost" algorithm which runs in the head of the gambler :

Algorithm

- "If I look at the entire set of races I've seen in my life, I can tell that it is very important that the other horses (other than the one I gamble on) will not be too strong. This is an important rule."
- "I will now concentrate on the races where my horse was the strongest horse, but nevertheless it didn't win. I recall that in some of these races, it was because there was a lot of wind, and even a strong horse might be sensitive to wind. This cannot be known in advance about a certain horse."
- "In another case the horse I chose was not the strongest, but even though, it won. This was because he was in the outer lane, and it is easier to run there."

Resulting strong classifier

- The other horses (other than the one I gamble on) should not be too strong. Importance of rule : 7 (on a scale of 1 to 10).
- These should not be wind in the day of the race. Importance : 3.
- The horse should be in the outer lane. Importance : 2.

Note that the importance of the rules are calculated according to their error in the execution of AdaBoost. this corresponds to the α of the rule in the algorithm (see table). Once the weak hypothesis h_t has been received from the weak learner, AdaBoost chooses a parameter α_t as in the table. Intuitively, α_t measures the importance that is assigned to h_t.

• Input : a sequence of N labeled examples $X = \langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ where $x_i \in X$ and $y_i \in Y = \{-1, 1\}$
integer T specifying number of iterations of AdaBoost.

• Initialize the weight vector : $w_i^1 = \frac{1}{m}$.

• Do for $t = 1, 2, \ldots, T$

1. Run the weak learner, providing it with the weights w_t.
The weak learner will return a simple classifier h_t
with $\Gamma(h_t) = \epsilon_t$.

2. Set the new weights vector :
$w_{t+1}^i \leftarrow \begin{cases} w_t^i \beta_t & \text{if } x_i \text{ is classified correctly by } h_t \\ w_t^i & \text{otherwise} \end{cases}$
where $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

3. Normalize the weights such that $\sum_{j=1}^{N} w_j^t = 1$.

• Output the classifier :

$$h(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)$$

where $\alpha_t = \log \frac{1}{\beta_t}$

TAB. 3.1 – The AdaBoost algorithm, in its original form

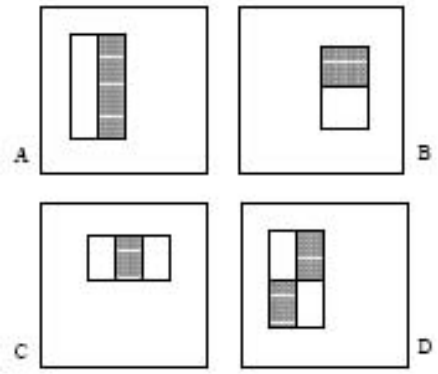


FIG. 3.7 – Example rectangle features shown relative to the enclosing detection window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Two-rectangle features are shown in (A) and (B). Figure (C) shows a three-rectangle feature, and (D) a four-rectangle feature.

Note that $\alpha_t \geq 0$ if $\epsilon_t < \frac{1}{2}$ (which we can assume without loss of generality), and that α_t gets larger as ϵ_t gets smaller. The distribution D_t is then updated using the rule shown in the figure. This rule actually increases the weight of examples misclassified by h_t , and decreases the weight of correctly classified examples. Thus, the weight tends to concentrate on "hard" examples.

The *final hypothesis* H is a weighted majority vote of the T weak hypotheses where α_t is the weight assigned to h_t .

3.2.4 Training a detector with AdaBoost

While the work of Papageorgiou et al presented in subsection 3.2.2 was innovative in terms of detection results, it left a substantial performance problem. The calculation of the wavelets demanded summing of pixels in a wide region, and this had to be done more than 1000 times for each examined subwindow! Of course, this could not be performed in something which comes near real-time processing, even when using specialized hardware.

The work of Viola and Jones [Viol 03] [Viol 01] brought a solution to this problem. Like Papageorgiou et al, Viola and Jones also use a set of features which are some version of Haar basis (though they also use related filters which are more complex than Haar filters, as we will see next). In order to compute these features very rapidly at many scales they introduce a new representation of an image called the *integral image*. The integral image can be computed from an original image using a few operations per pixel. Once computed, any one of these Haar-like features can be computed at any scale or location in constant time (not dependent of the size of the rectangle of the "wavelet").

Their second contribution is a method for constructing a classifier by selecting a small number of important features using AdaBoost. Recall that Papageorgiou et al chose their subset of 29 wavelet-like features manually.

If we consider all possible wavelet-like features (in all scales and all directions, as seen

in figure 3.7), then within any image subwindow the total number of these features is very large, far larger than the number of pixels. In order to ensure fast classification, the learning process must drop a large majority of the available features, and focus on a relatively small set of critical features. The solution of Viola and Jones for this feature selection is achieved by the AdaBoost algorithm : the weak learner is an algorithm that searches for a weak classifier consisting on only a single feature (this idea is originated in [Tieu 99]). As a result of this method, each stage of the boosting process, which selects a new weak classifier, is actually a "feature selection" process. AdaBoost is good for this purpose because it provides high level of learning - that is, the results are well generalized on new data (some proofs of this are given in [Scha 97],[Osun 97],[Papa 98b]).

The third major contribution of Viola and Jones is a method for combining successively more complex classifiers in a "cascade" structure which dramatically increases the speed of a detector by focusing on "promising" regions of the image. The idea behind the focus of attention is that it is often possible to rapidly determine where in an image an object might occur (see results for this hypothesis in [Amit 97]|[Itti 98]|[Tsot 95]). Once "easy" parts are dropped, more processing is performed only for the regions which are left.

To make this approach work, it is very important that the "false negative" rate of such step in this process be very low. In other works, almost all objects should be selected by each layer of the detection process ; if one of them drops the object, it will not be detected in the final result.

The training process of the "cascade" is done layer after layer, when each layer is trained as a simple AdaBoost, as described in the previous section.

For a good example of the process, we give some details from the domain of face detection, where Viola and Jones actually started their research of the "cascade" (only in a later work they used pedestrians). In this case, they show that it is possible to achieve fewer than 1% false negatives (that is, 99% of the positive cases remain) and 40% false positives using a classifier constructed from only two Haar-like features.

The effect of such a simple filter is to reduce by over one half the number of locations where the final detector must be evaluated. Those sub-windows which are not rejected by the initial classifier are processed by a sequence of classifiers, each slightly more complex than the last. If any classifier rejects the sub-window, no further processing is performed.

The rest of this subsection is describing the framework of detection with AdaBoost and the use of the integral image. The next subsection is giving some more details about the attentional cascade.

Extended wavelet-like features

The object detection procedure of Viola and Jones classifies images based on the value of simple features. These features are based on the "Haar basis" functions which have been used by Papageorgiou et al. However, here they are expended a little bit.

Viola and Jones use three kinds of features. The value of a two-rectangle feature is the difference between the sum of the pixels within two rectangular regions. The regions have the same size and shape and are horizontally or vertically adjacent (see Figure 3.7). A three-rectangle feature computes the sum within two outside rectangles subtracted from the sum in

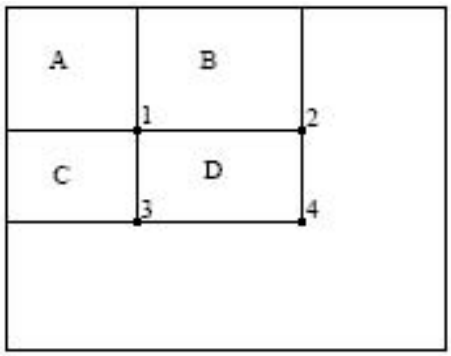


FIG. 3.8 – The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value of the integral image at location 2 is the sum of the pixels in rectangles A and B. The value of the integral image at location 3 is the sum of the pixels in rectangles A and C. The value of the integral image at location 4 is the sum of the pixels in rectangles A,B,C and D. Therefore 4+1-2-3=(A+B+C+D)+A-(A+B)-(A+C)=D.

a center rectangle. Finally a four-rectangle feature computes the difference between diagonal pairs of rectangles. In the case of face detection, they used a base resolution of 24x24. With this resolution, the size of the entire set of rectangle features is around 180,000. We say that this set of rectangle features is overcomplete (unlike the Haar-basis) because there are elements which can be achieved by summing other elements (the same meaning as in a basis of numbers in a space, in algebra).

The integral image

As already mentioned above, the rectangle features can be computed very rapidly using a new representation for the image which they call the integral image. The integral image at location x, y contains the sum of the pixels above and to the left of x, y , inclusive :

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{3.14}$$

where $ii(x, y)$ is the integral image and $i(x, y)$ is the original image. The integral image can be calculated from an input image in one pass (trivial).

Using the integral image any rectangular sum can be computed in four array references (see Figure 3.8). Therefore the difference between two rectangular sums can be computed in eight references. However since the two-rectangle features defined above have adjacent rectangular sums, they can be computed in six array references. In the same way we can see that we need eight references in the case of the three-rectangle features, and nine for four-rectangle features. As this seem very little, in further chapters in this thesis we will see that we develop features which need around 4 image references (in average) and do not need the preparation of the integral image.

AdaBoost usage

Given a feature set and a training set of positive and negative images, Viola and Jones chose to use AdaBoost, as already mentioned. Recall that there are over 180,000 rectangle features in each image sub-window, more than the number of pixels. Even though each feature can be computed very efficiently, computing the complete set is too difficult. Viola and Jones followed the feeling that Papageorgiou already had, that a small number of features is enough. But instead of selecting it manually (as done by Papageorgiou) they used AdaBoost.

While running AdaBoost, the weak learning algorithm is designed to select the single rectangle feature which best separates the positive and negative examples (it runs on all the 180,000 possibilities). For each feature, the weak learner determines also the optimal threshold value, such that the minimum number of examples are misclassified (according to the weights given to it by AdaBoost). A weak classifier $h_j(x)$ thus consists of a feature f_j , a threshold θ_j and a parity p_j indicating the direction of the inequality sign :

$$h_j(x) = \{ \begin{array}{ll} 1 & \text{if } p_j f_j(x) < p_j \theta_j \\ 0 & \text{otherwise} \end{array} \tag{3.15}$$

In this case x is a sub-window of an image[3]. See Table 3.2 for a summary of the boosting process.

Of course, no single feature can perform the classification task with low error. It is interesting however to see the errors of the selected simple features learned by AdaBoost : features which are selected in early rounds of the boosting process have error rates between 0.1 and 0.3. Features selected in later rounds, as the task becomes more difficult, have error rates between 0.4 and 0.5.

The Cascade

This subsection gives more details about the other major contribution of Viola and Jones, namely an algorithm for building a "cascade" of classifiers which achieves about the same detection performance as "flat" AdaBoost, and dramatically reduces computation time. As mentioned above the general idea is that smaller (and more efficient) strong classifiers can be built and reject many of the negative sub-windows, and still detect almost all positive instances. This can be achieved by "playing" with the threshold of a boosted classifier, so that the false negative rate is close to zero. After simpler classifiers are used to reject the majority of subwindows, more and more complex classifiers are used to achieve low false positive rates.

The overall form of the detection process is that of a degenerate decision tree (see Figure 3.8). Each classifier is activated for a given sub-window if the classifier before it answered positively. If one of the classifiers answers negatively, the sub-window is dropped.

The "layers" in the cascade are constructed by training classifiers using AdaBoost as described in previous subsection, and then adjusting the threshold (the number which is usually 0.5) to minimize false negatives rate. Note that the default AdaBoost threshold (0.5)

[3]as mentioned already Viola and Jones used 24x24 window for face detection. In later work they used 16x24 for pedestrian detection

• Input : a sequence of N labeled examples $X = \langle (x_1, y_1), \ldots, (x_N, y_N) \rangle$ where $y_i \in \{0, 1\}$ integer T specifying number of iterations of AdaBoost.

• Initialize the weight vector : $w_i^1 = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negative and positive examples, respectively.

• Do for $t = 1, 2, \ldots, T$

1. Normalize the weights such that $\sum_{j=1}^{N} w_j^t = 1$.

2. Run the weak learner, providing
   it with the error function $\Gamma(h) = \sum_{i=1}^{N} w_i^t \mid h(x_i) - y_i \mid$
   The weak learner will return a simple classifier h_t
   with $\Gamma(h_t) = \epsilon_t$.

3. Set the new weights vector :
   $w_{t+1}^i \leftarrow \begin{cases} w_t^i \beta_t & \text{if } x_i \text{ is classified correctly by } h_t \\ w_t^i & \text{otherwise} \end{cases}$
   where $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

• Output the classifier :

$$h(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq 1/2 \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

TAB. 3.2 – The AdaBoost algorithm, in the same way used by Viola and Jones

is designed to give a low error rate on the training data and not specifically low false negative. So in general, layers of the cascade have a lower threshold, in order to yield higher detection rates (i.e. lower false negative rate) and higher false positive rates.

In the training process, each classifier is trained using the examples which passed all the previous classifiers. So in the training process, the classifier has to solve a "harder" problem than the previous classifier. More formally, the training process consists of the steps shown in table 3.3.

Note that during this process, some layers will not need any features in order to achieve their "task". On the other hand, it can happen that a layer will add more and more features and will never reach its goal. This means that the input values were too hard.

Using motion information

A later work of Viola and Jones describes a pedestrian detection system that integrates intensity information with motion information. In the work described here, Viola and Jones use information about motion as well as intensity information. The implementation detects pedestrians at small scales (the base size is 20x15 pixels).

The system is calculating short term motion, based only on the difference between an image and the previous image. Theoretically, they try to create one system which will perform the detection and the tracking. We will review the system and later see its advantages and disadvantages.

General framework

The dynamic pedestrian detector that Viola and Jones built is based on the simple rectangle filters presented in the previous subsection. Here it will be described how they extend these filters to act on motion pairs.

Viola and Jones describe a generalization of their original features by working with the differences between consecutive pairs of images. Of course, some information about motion can be extracted from these differences, and in the case of pedestrians (cyclic movement of legs and hands) it seems quite promising. To improve results, information about the direction of motion can be extracted by calculating the difference between the second image *after shifting* and the first image. Formally, the motion filters used by Viola and Jones work on 5 images :

$$\Delta = abs(I_t - I_{t+1}) \tag{3.16}$$

$$U = abs(I_t - I_{t+1} \uparrow) \tag{3.17}$$

$$L = abs(I_t - I_{t+1} \leftarrow) \tag{3.18}$$

$$R = abs(I_t - I_{t+1} \rightarrow) \tag{3.19}$$

$$D = abs(I_t - I_{t+1} \downarrow) \tag{3.20}$$

- Input :
  1. a positive examples set S_p
  2. a large set of negative examples $\widehat{S_n}$ (in the order of millions of examples ; in our work, we typically took all the sub-windows of a video sequence not containing the object)
  3. desired detection rate $0 < t < 1$ (typical example : 0.85)
  4. desired false positive rate $0 < f < 1$ (typical example : 0.000001).
  5. desired number of layers L
  6. Validation set V combined of negative and positive examples. To achieve good training, this set must be large. Typically we use several hundreds of positive and several millions of negatives.
- Initialize an empty classifier H_0 (answering "yes" for every example it gets as input)
- We denote by $\hat{t}_l = t^{l/L}$ the desired detection rate of the detector made of layers $1 \dots L$.
- We denote by $\hat{f}_l = f^{l/L}$ the desired false positive rate of the detector made of layers $1 \dots L$.
- for $l = 1 \dots L$ do :
  1. Prepare a set $S_n \subset \widehat{S_n}$ made of 10000 examples which are positively classified by H_{l-1}. These are, actually, the false detections of all the previous layers. We want to concentrate on them.
  2. Run normal AdaBoost training to train layer l, in the following way :
     - Each time, before adding another feature to the layer l, test the whole detector (made of layers $1 \dots l-1$ and the part already existing of layer l) on the validation set V.
     - Tune the threshold of layer l until the detection rate on the validation set becomes higher/equal than $\hat{t}_l$. This will affect the false positive rate.
     - After having found the threshold that provides a detection rate higher/equal than $\hat{t}_l$, check the false positive rate. If it is smaller than $\hat{f}_l$, continue to add layers. If not, the layer is ready.

TAB. 3.3 – The cascade training process

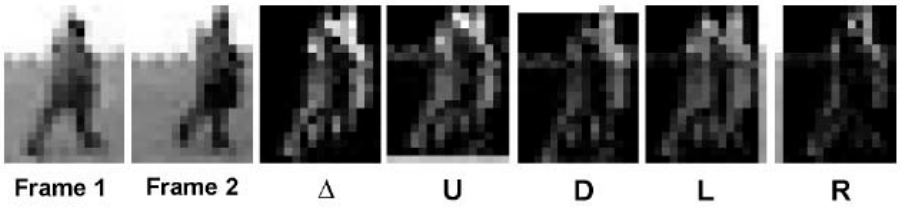


FIG. 3.9 – A close look on 20x15 pixel images, an example of the various shifted difference images used in the algorithm of Viola and Jones. The first two images are two typical frames with a low resolution pedestrian. The following images show the Δ, U, D, L and R images described in the text. Notice that the right-shifted image difference (Δ), which corresponds to the direction of motion shown in the two frames, has the lowest energy.

where I_t and I_{t+1} are images in time, and $\{\uparrow, \leftarrow, \rightarrow, \downarrow\}$ are image shift operators (for example, $I_t \uparrow$ is I_t shifted up by one pixel). See Figure 3.9 for an example of these images.

The filters themselves are as follows :

$$f_i = r_i(\Delta) - r_i(S) \tag{3.21}$$

$$f_j = \phi_j(S) \tag{3.22}$$

$$f_k = r_k(S) \tag{3.23}$$

$$f_m = \phi_m(I_t) \tag{3.24}$$

where S is one of U, L, R, D and $r_i()$ is a single box rectangular sum in the detection window. ϕ_i is one of the rectangle filters shown in figure 3.7.

f_i compares sums of absolute differences between Δ and one of U, L, R, D. f_j compares sums within the same motion image. f_k measures the magnitude of motion in one of the motion images. f_m are the static filters described in the previous sections, which are simply rectangle filters that work on the first input image, I_t.

Because the filters shown in figure 3.7 can have any size, aspect ratio or position as long as they fit in the detection window, there are very many possible motion and appearance filters. The learning algorithm selects features from this large number of filters and builds the best classifier. A classifier, C, is a thresholded sum of features :

$$C(I_t, I_{t+1}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{N} F_i(I_t, \Delta, U, L, R, D > \theta) \\ 0 & \text{otherwise} \end{cases} \tag{3.25}$$

where a feature F is a thresholded filter that outputs one of two values :

$$C(I_t, I_{t+1}) = \begin{cases} \alpha & \text{if } f_i(I_t, \Delta, U, L, R, D > t_i) \\ \beta & \text{otherwise} \end{cases} \tag{3.26}$$

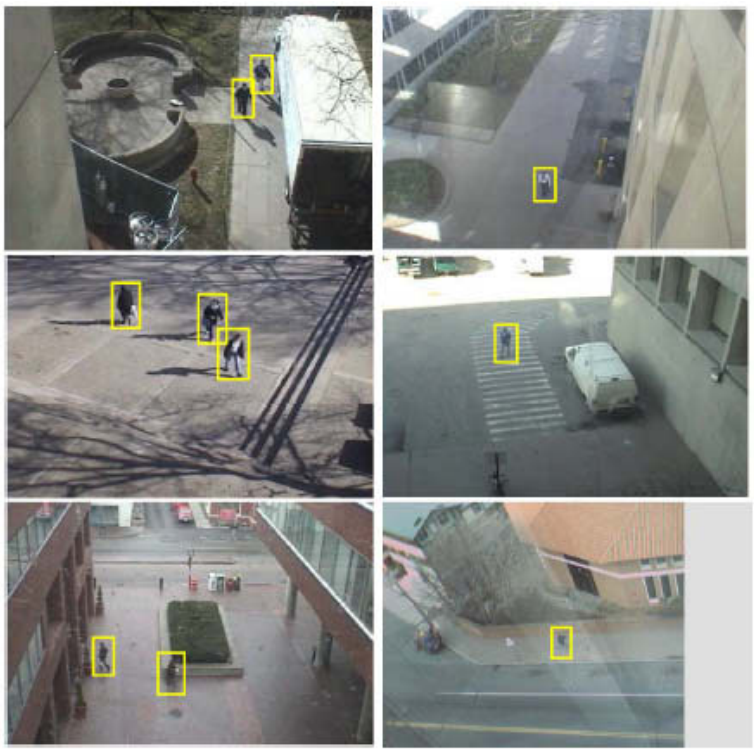


FIG. 3.10 – Sample frames from each of the 6 sequences we used for training.

where $t_i \in \Re$ is a feature threshold and f_i is one of the motion or appearance filters defined above. The real values α and β are computed during AdaBoost learning, and also the filter itself, the filter threshold t_i, and the classifier threshold θ.

Experimental data and the training process

For working with the system, the authors created a set of 8 video sequences of street scenes with all pedestrians marked with a box in each frame. Each sequence contained around 2000 frames. 6 sequences were used for training (see Figure 3.10) and 2 for testing.

In the learning process, they learned both a dynamic pedestrian detector and a static pedestrian detector. The dynamic detector was trained on consecutive frame pairs and the static detector was trained on single frames. The static pedestrian, of course, did not use the motion filters but only the static set of filters (denoted f_m above).

Each stage of the cascade is a boosted classifier trained using a set of 2250 positive examples and 2250 negative examples. Each positive training example is a pair of 20 x 15 pedestrian images taken from two consecutive frames of a video sequence (or a single image, for the static detector). Negative examples are similar image pairs of non-pedestrians images. During training, the variance of all example images is normalized to reduce contrast variations. Of course, the same variance normalization is performed when testing a new image.

In figure 3.11, we see the first 5 features learned for the dynamic detector. We see that the detector is using the motion information. In figure 3.12, we see the first 5 features learned

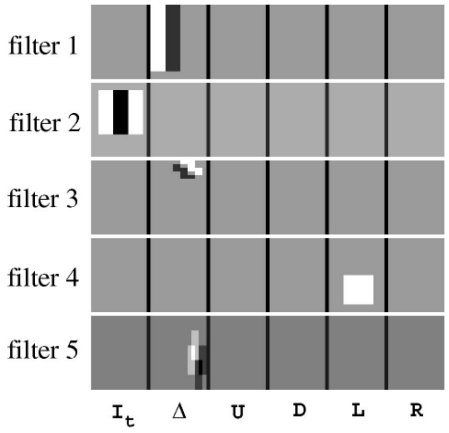


FIG. 3.11 – The first 5 filters learned for the dynamic pedestrian detector. The 6 images used in the motion and appearance representation are shown for each filter.

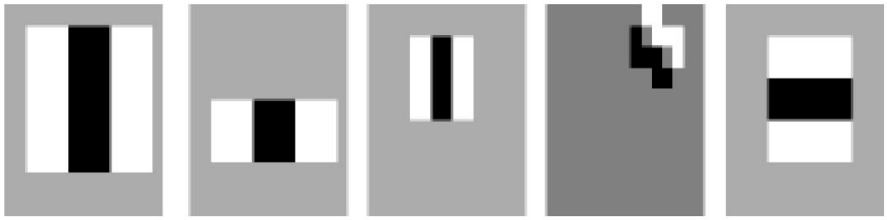


FIG. 3.12 – The first 5 filters learned for the static pedestrian detector.

for the static detector.

Results

The authors ran both the static and dynamic detectors over test sequences for which they created ground truth. The resulting ROC curves are shown in figures 3.13 and 3.14. The ROC curve for the dynamic case is much better on sequence 2 than the static case. On sequence 1, the dynamic detector is only slightly better than the static one.

3.3 The control-points features

One notable problem in all the works presented in previous sections is the need to normalize the images before passing them to the classifier. Wavelet-based features are based on thresholding sums of pixel values. This implies strong dependency on image intensity, and this is why these features need histogram stretching or equalization prior to classifying. Without this prior normalization, results dramatically deteriorate [Evgc 00]. Figure 3.15 demonstrates how wavelet-based features focus primarily on intensity and not on shapes.

Normalization operations dramatically reduce the overall performance of the system. In addition, one can never be sure that such an operation performs well, since a single pedestrian

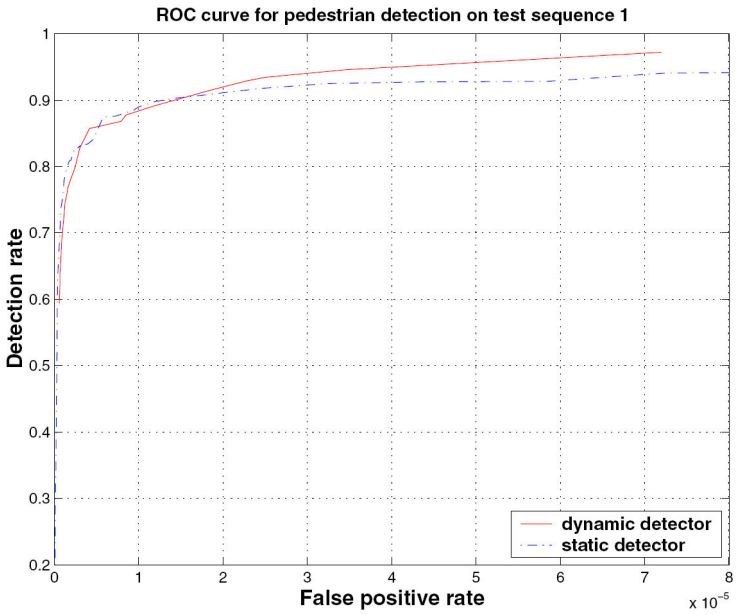


FIG. 3.13 – ROC curve on test sequence 1 for both the dynamic and static pedestrian detectors.

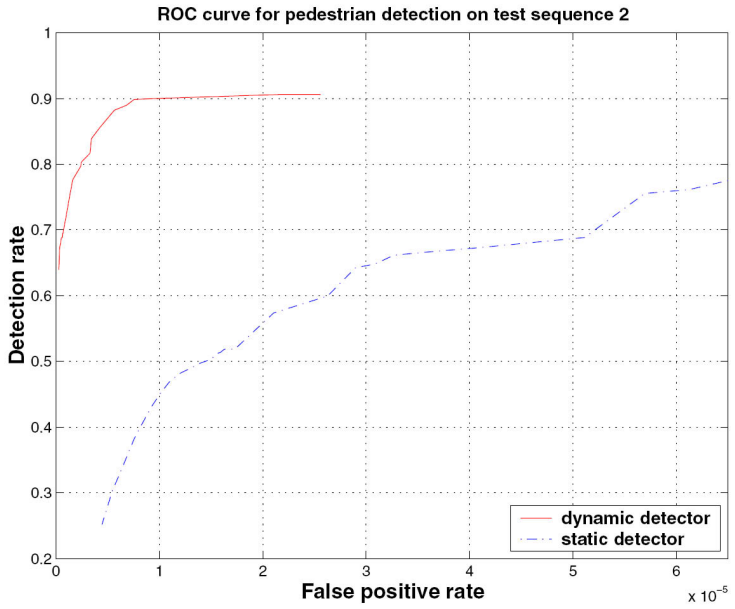
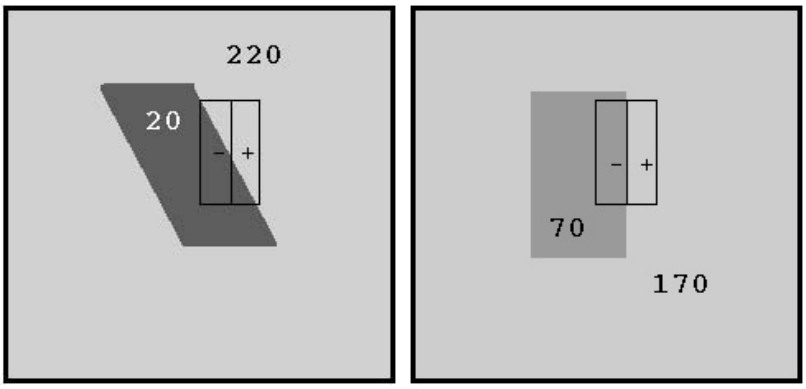


FIG. 3.14 – ROC curve on test sequence 2 for both the dynamic and static pedestrian detectors.



(a) (b)

FIG. 3.15 – The intensity problem in wavelet-based features. In both images, we apply a simple wavelet feature which calculates the difference between the sum of pixels found in the rectangle marked by + and the sum of pixels found in the rectangle marked by -. One can observe that the feature yields approximately the same value for both images, even though the shapes presented in the images are not the same. This example explains why the performance of this kind of features dramatically depends on prior normalization of the image.

can have different intensities in his different parts. The features we define in this paper are intensity independent, because they do not use numerical summing of pixel values.

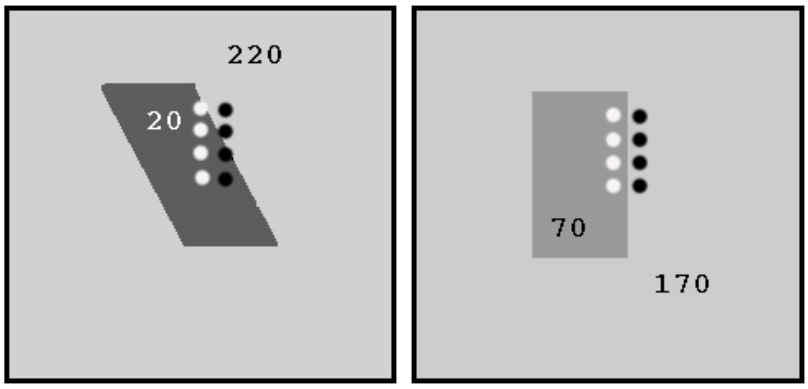
Another issue that needs to be mentioned in this context is the use of dynamic information. Papageorgiou et al show moderate improvement in detection rate when using temporal information [Papa 99]. Viola and Jones show excellent results in their dynamic pedestrian detector but rather poor results in the static one [Viol 03]. When considering a system with an on-board camera, using dynamic information is problematic. Camera movements add a large amount of noise, which is not always easy to eliminate. From the application side, it should be reminded that pedestrian detection and impact prediction is the most important when the car is moving.

The features we present in this subsection work faster than the features of Viola and Jones, with much less preparations that have to be done on the image. They work on static, gray images.

3.3.1 Independency of illumination

Given an image of width W and height H, we define a *control point* to be an image location in the form $\langle i, j \rangle$, where $0 \leq i < H$ and $0 \leq j < W$. Given an image location i, we denote by $val(i)$ the pixel value in that location.

Our feature consists of two sets of control points, $x_1 \dots x_n$ and $y_1 \dots y_m$. The sizes of the sets are bounded : $n, m \leq \Phi$, where the optimal value of Φ is discussed later. The feature contains also a threshold value T and a working resolution R of the image, which can be either $48x24$, $24x12$ or $6x12$.



(a) (b)

FIG. 3.16 – The intensity-independent features. The feature answers "yes" if and only if the difference between *every* black control point and *every* white control point is larger than a certain threshold (the feature is symmetric ; black and white can be switched). This verifies that only images which have a certain shape (in this case, a vertical border) will be positively classified. When such a shape is absent (as in figure (a)), the feature will answer negatively, regardless of the intensity of the pixels.

To classify a given image, we first scale the image to the resolution R. Then, the feature answers "yes" if and only if for every control point $x \in x_1 \dots x_n$ and every control point $y \in y_1 \dots y_m$, the value $\|val(x) - val(y)\|$ is larger than T. This is demonstrated in figure 3.16.

One can immediately notice that for computing the result, a good implementation does not have always to check all the $m+n$ control points. The calculation can be stopped once the condition of the feature is broken, and this usually happens much before all the control points are checked. In later sections we show that the average number of control points checked is around 4. Thus, this feature demands even less operations than the features of Viola and Jones *without* preparing an integral image.

One can also argue that this type of features is too sensitive to noise, because it relies on the values of single pixels and not of regions (with one exception, that using the quarter and sixteenth resolution is somewhat similar to checking regions). While this is true, the fact remains (as it is shown later in this document) that these features achieve similar detection rates in less operations, comparing to the wavelet-based features. In section 3.3.4 we further discuss the issue of sensitivity to noise.

To achieve high detection speed, we work only on gray images. While this reduces the detection rate, it is essential for obtaining real-time operation with this feature.

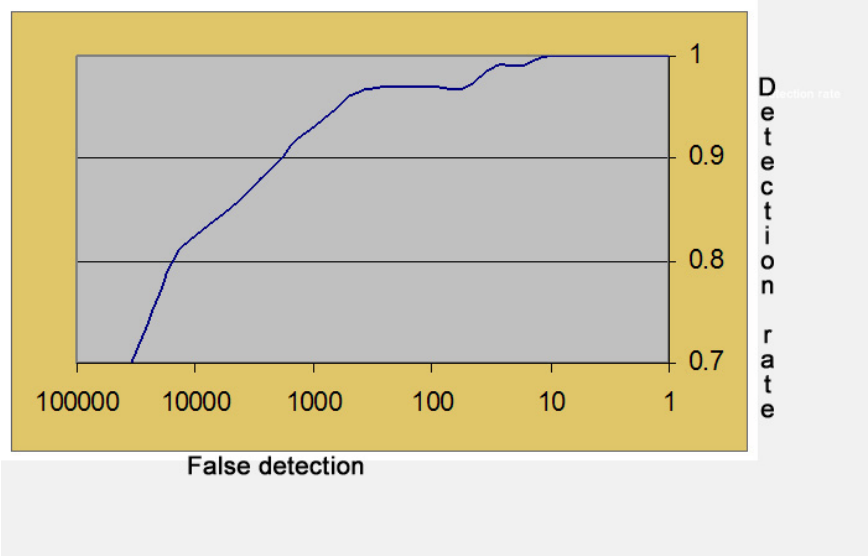


FIG. 3.17 – ROC curve of the illumination-independent feature on the MIT set.

FIG. 3.18 – Successful detections in outdoor images. One can see that the feature operates equally at night and day, without any normalization of the image.

3.3.2 Initial experiments

With the MIT pedestrian images

We have tested our new features using AdaBoost and a the 924 pedestrian images used in [Oren 97]. We used 824 images for training and 100 for testing. As negative examples we used 7310 (for the learning) and about 1 million (for the testing) sub-windows from outdoor images not containing pedestrians. Using $\Phi = 6$ we ran 1320 AdaBoost steps after verifying that no overfitting occurs. The resulting strong classifier was tested on the test data, obtaining the ROC curve shown in figure 3.17. One can see that results are similar to the ones obtained by Papageorgiou et al [Oren 97], using gray images and 1326 wavelet features trained with SVM.

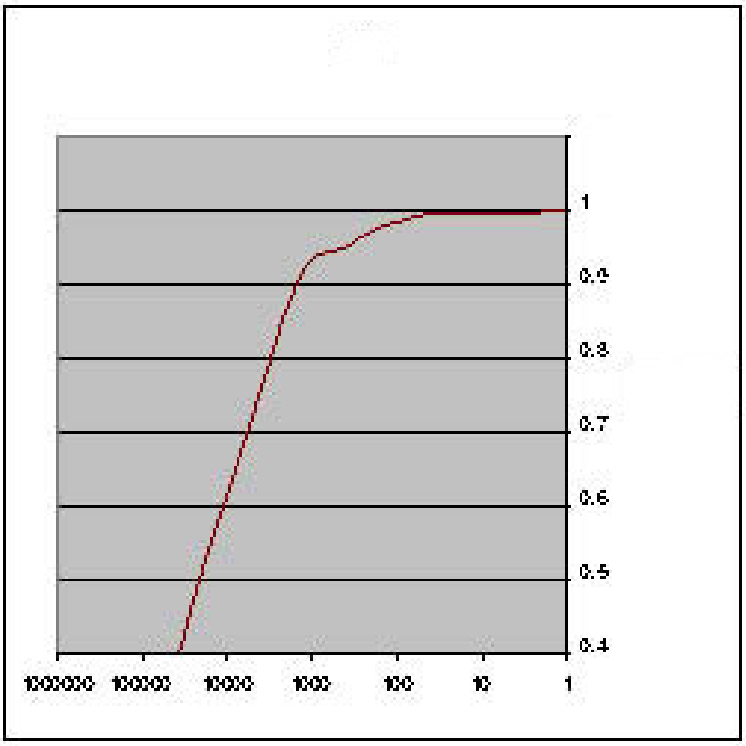


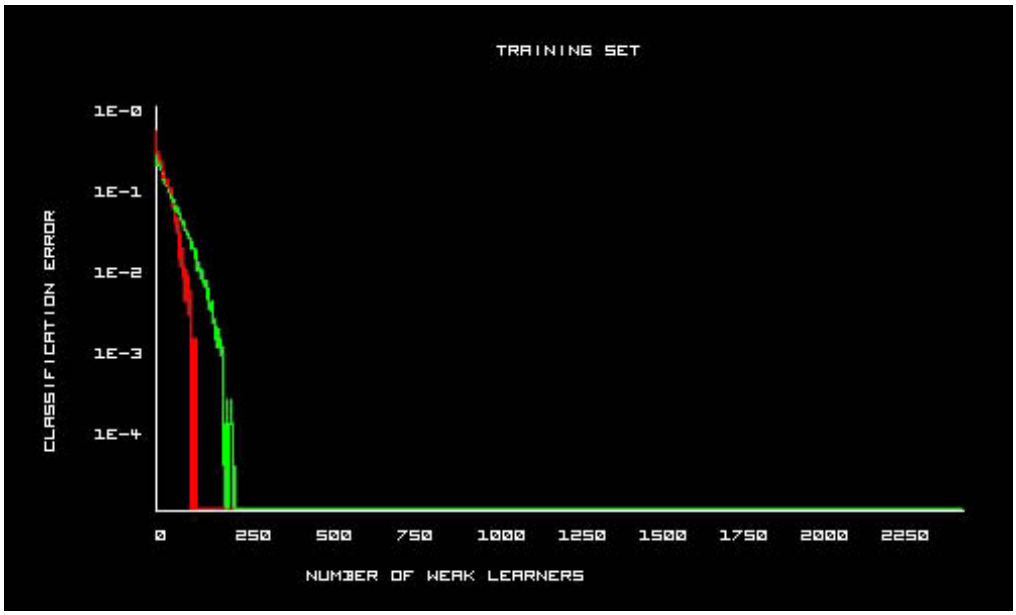
FIG. 3.19 – ROC curve of the illumination-independent features on the SEVILLE data set

With the SEVILLE pedestrian images

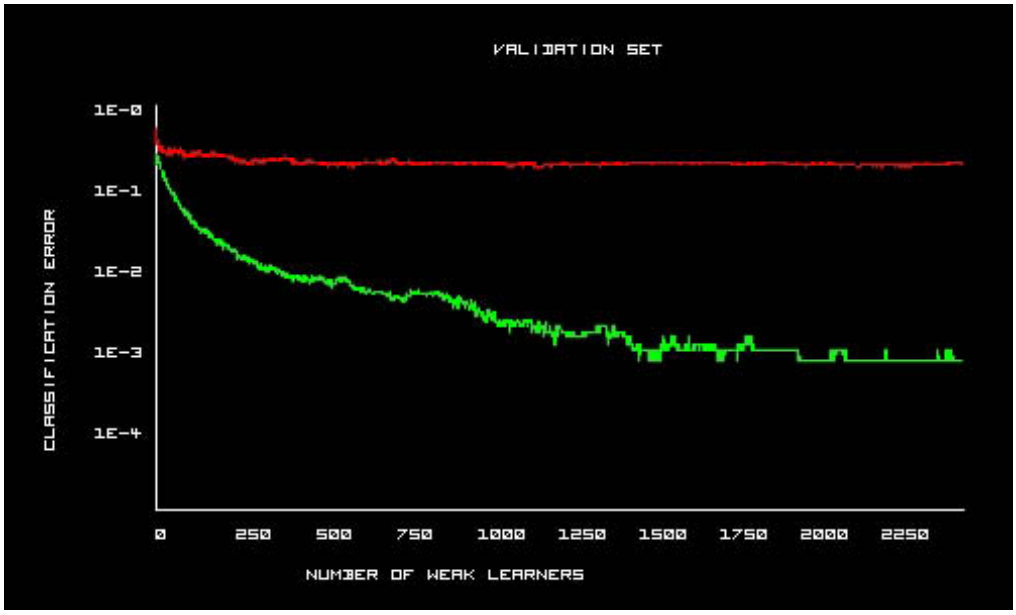
We also tested the feature on the data set created by the SEVILLE program, which is described in section 3.7. The SEVILLE data set contains 1183 positive examples (24x48 pedestrian images) and 13175 negative examples (24x48 images not containing pedestrians). 1/3 of these images were left aside as a validation set, and the remaining 2/3 were used for the learning. Then, results are tested on the validation set. These results are reliable, since the examples in the validation set were not used in the training process.

Figure 3.19 shows the ROC curve on the validation set with 2000 weak rules. Note that the SEVILLE set is much harder than the MIT data set. Here, resolution is smaller, and the pedestrians are appearing in different angles, not just forward and backward facing pedestrians.

Figure 3.20 shows the learning graph on the validation set and the training set. The error on the positive and negative examples is evaluated as a function of the number of weak learners. Obviously the learning set is well adapted after a small number of weak rules. From observation of the validation set graph, it is clearly seen that after 2000 weak rules, there is no more improvement.



(a)



(b)

FIG. 3.20 – Learning graph of the illumination-independent features on the SEVILLE data set - (a) learning set, (b) validation set.

3.3.3 Genetic algorithm as a weak learner

The AdaBoost algorithm needs a weak learner - an algorithm which will provide a "good" feature each time, when the goodness is measured according to the current weights of the examples. Obviously, choosing the best simple classifier at each step of AdaBoost cannot be done by testing all the possibilities, since there are about 10^{36} of them. Therefore, a genetic-like algorithm is used, which starts with a set of random simple classifiers and iteratively improves them while using the following mutations :

- *RemoveControlPoint* chooses a control point and removes it from the feature.
- *AddControlPoint* adds a control point with random location to the feature.
- *MoveControlPoint* chooses a control point and changes its location.

The genetic-like algorithm, given in table 3.4, maintains a set of simple classifiers which are initialized as random ones. Each step of the algorithm a new "generation" of simple classifiers is produced by applying the 3 types of mutations on each of the simple classifiers. All 3 mutations are tested and the one with the lowest error might replace the "parent" if it has a lower error. In addition, some random simple classifiers are added at each step.

The genetic algorithm continues to evolve the generations until there is no improvement during 40 consecutive generations. An improvement for that matter is of course a reduction in the error of the best feature in the generation.

Figure 3.5 shows a snapshot from the operation of the AdaBoost and the genetic algorithm. Here are some explanations :

At the beginning of the process it is indicated that we are using features with a maximum of 6 control points and pedestrian dimensions of 24x48. Next the examples set is divided into validation and training set.

Next, training examples are used for the learning. AdaBoost is started, and weights are uniformly assigned to the examples. Then, the genetic algorithm is called, generating a first generation whose best feature has an error of about 36% (with 50 positive examples out of 59 well classified and 204 out of 466 negative examples well classified). Next generation contains already a better feature (23%, 43/59 pos, 378/466 neg) and more generations are produced. At some stage, there is no improvement after 41 consecutive generations, and the process stops. AdaBoost then continues with more steps.

In the exert, two more AdaBoost steps are executed, obtaining a strong classifier made of 3 weak rules.

The validation phase of the process is checking the final strong classifier with respect to the training set and the validation set. See section 3.7 for details and graphs.

3.3.4 Feature discussion

Figure 3.21 shows several interesting features that were learned in the training process. Figure 3.18 shows successful detections on new images obtained by scanning all possible sub-windows. It can be seen that the same detection quality is obtained in all lighting conditions.

• Input : an error measurement Γ that matches an error to every simple classifier, a number G of generations to run, and a number N of "survivors" at each generation.

• Let Υ be a generator of random simple classifiers. Initialize the first generation's simple classifiers vector C^1 as :
$c_i^1 \leftarrow \Upsilon \qquad i = 1 \ldots N$

• Do for $g = 1, 2, \ldots, G$:

1. Build the next generation vector C^{g+1} as :
   $c_i^{g+1} \leftarrow c_i^g \qquad i = 1 \ldots N$
2. Do for $m = 1, 2, 3$:
   if $\Im^m(c_i^g)$ is valid and $\Gamma(\Im^m(c_i^g)) < \Gamma(c_i^{g+1})$ then
   $c_i^{g+1} \leftarrow \Im^m(c_i^g) \qquad i = 1 \ldots N$
3. Enlarge the vector C^{g+1} with N additional simple classifiers, chosen randomly :
   $c_i^{g+1} \leftarrow \Upsilon \qquad i = N+1, \ldots, 2N$
4. Sort the vector C^{g+1} such that
   $\Gamma(c_1^{g+1}) \leq \Gamma(c_2^{g+1}) \leq \ldots \leq \Gamma(c_{2N}^{g+1})$

• Output the simple classifier c_1^{G+1}.

A simple classifier is *valid* if it consists of a valid image region (i.e. with a positive surface and not exceeding image borders) and of a threshold in the range $[0, 255]$.

TAB. 3.4 – The genetic-like algorithm.

```
Starting, MAX_CONTROL_POINTS=6, dimensions (24,48)
Dividing the examples: 2/3 training, 1/3 validation
Division file exists in ..\..\PedesData\simpleSet\/positive, not dividing.
Division file exists in ..\..\PedesData\simpleSet\/negative, not dividing.
Division file exists in ..\..\PedesData\simpleSet\/nolabel, not dividing.
....Loaded direcory 27 positive
.............................Loaded direcory 257 negative
Loaded direcory 0 unlabled
Read validation images (27 positive, 257 negative)
.....Loaded direcory 59 positive
.....................................Loaded direcory 466 negative
Loaded direcory 0 unlabled
Read training images (59 positive, 466 negative)


**** Start training
Initializing weights...
Starting AdaBoost run.
AdaBoost step 0
Minimum error = 0.357387, 0th time (50/59 pos, 204/466 neg)
Minimum error = 0.230014, 0th time (43/59 pos, 378/466 neg)
Minimum error = 0.206209, 0th time (51/59 pos, 337/466 neg)
Minimum error = 0.179275, 0th time (50/59 pos, 370/466 neg)
Minimum error = 0.179275, 1th time (50/59 pos, 370/466 neg)
Minimum error = 0.160617, 0th time (56/59 pos, 340/466 neg)
Minimum error = 0.158453, 0th time (46/59 pos, 421/466 neg)
Minimum error = 0.080036, 0th time (55/59 pos, 423/466 neg)
Minimum error = 0.080036, 1th time (55/59 pos, 423/466 neg)
Minimum error = 0.080036, 2th time (55/59 pos, 423/466 neg)
Minimum error = 0.074671, 0th time (55/59 pos, 428/466 neg)
Minimum error = 0.074671, 1th time (55/59 pos, 428/466 neg)
Minimum error = 0.074671, 2th time (55/59 pos, 428/466 neg)
Minimum error = 0.070488, 0th time (56/59 pos, 424/466 neg)
...
Minimum error = 0.025533, 40th time (57/59 pos, 458/466 neg)
Minimum error = 0.025533, 41th time (57/59 pos, 458/466 neg)

AdaBoost step 1
Minimum error = 0.359515, 0th time (31/59 pos, 402/466 neg)
...
Minimum error = 0.008753, 41th time (58/59 pos, 458/466 neg)

AdaBoost step 2
Minimum error = 0.245077, 0th time (56/59 pos, 242/466 neg)
...
Minimum error = 0.013993, 41th time (53/59 pos, 463/466 neg)

** Validation phase **
Begin of AdaBoost ROC analyze for training set...
Filling classification matrix....
Performing steps analysis......
Performig ROC analysis.........................................
End of ROC analysis.
Begin of AdaBoost ROC analyze for validation set...
Filling classification matrix....
Performing steps analysis......
Performig ROC analysis.........................................
End of ROC analysis.
```

TAB. 3.5 – The operation of the genetic-like algorithm as a weak learner.

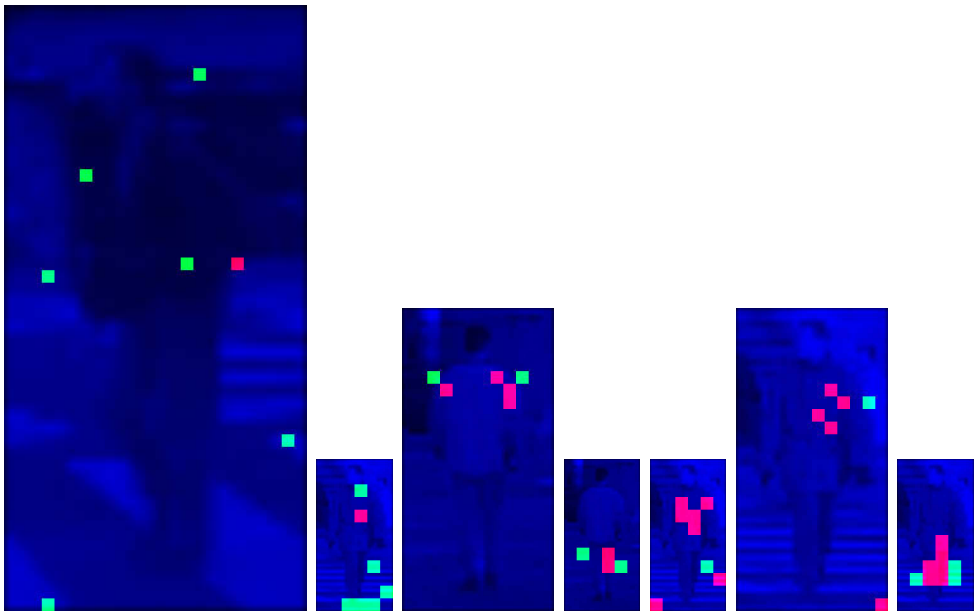


FIG. 3.21 – Some examples of features learned in different stages of the training process. The two sets of control points are shown in green and red. The input image is tested in one of three resolutions.

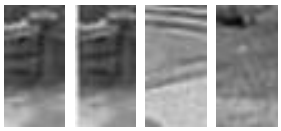


FIG. 3.22 – Some examples of false positives. One can see these are not conventional.

Sensitivity to noise

Since this feature is checking the value in single pixels and not in regions, it is expected to be more sensitive to noise. Rather surprisingly, this sensitivity is well hidden by the boosting process and the strong classifier resulting from AdaBoost is yielding good results, as seen before. However, it is interesting to look at some false positives obtained by the classifiers, as shown in figure 3.22. One can see that these are not "conventional" false positives. They are far from visually resembling a pedestrian. This implies, that a combination with a conventional, simple classifier (like a simple vertical edges detector, or a single wavelet-like feature) might improve results.

When working with video sequences, another interesting phenomenon is observed : as opposed to conventional false positives, these ones are more chaotic ; they are less likely appear in the same area in consecutive images. Thus, they might be easier to cope with than "logic" false positives. One can think of a simple temporal filter that validates a pedestrian target only if it appears in the same area of the image in 2-3 consecutive images, thus easily removing false positives coming from noise. Such a filter would have much harder work eliminating false positives originating from real pedestrian-like shapes, because they might repeat themselves in consecutive images.

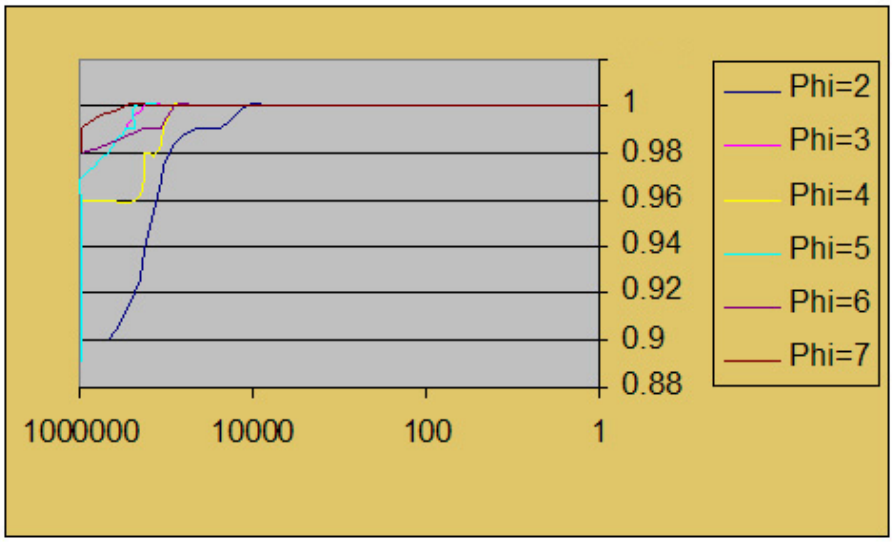


FIG. 3.23 – The ROC curve on a test set with various values of Φ.

Selection of the maximal number of control points

In section 3.3.1 we mentioned that the sizes of the two sets of control points are bounded by a number Φ. In this section we discuss how to choose the right Φ. Obviously we can expect that a higher Φ would yield better detection rate, but also slower operation. Figure 3.23 shows the ROC curves obtained on the test data using classifiers that were trained with different values of Φ. On the other hand, table 3.6 shows the average number of image access operations needed for the classifiers which were trained with these different values. One can see that substantial improvement is achieved until $\Phi = 6$ with moderate performance cost. According to this information, a value of $\Phi = 6$ seems like a good choice.

It is interesting to see that on $\Phi = 7$ the detection rates fall down a bit. This phenomenon has a possible explanation related to the training process : since we are not checking for all possibilities of the learning process (in each AdaBoost step) but use various searching techniques, a larger Φ implies a larger search space. This lowers the chances to find a good feature.

To understand this point, consider again the genetic algorithm described in table 3.4. We can see, that the search advances using mutations (see step 2 of the algorithm) ; in such a way, if a good solution is found, the algorithm tries to find an even better one in its neighborhood. At the same time, the algorithm keeps searching randomly for new features, using randomization (see step 3 of the algorithm). It is clear, that the larger the space is, the lesser the chance to "fall" randomly on a feature that is better than the best one found so far.

3.3.5 Using the attentional cascade

The results given in the previous sub-sections were used to construct a real-time pedestrian detection application. The application uses the illumination independent features and Viola

TAB. 3.6 – The average number of operations needed to calculate the feature as a function of Φ, the upper bound on the number of control points.

value of Φ	Average number of operations
2	3.306
3	3.624
4	3.781
5	4.014
6	4.061
7	4.112

and Jones' attentional cascade.

Training a cascade is a long process, as described in [Viol 01]. After each AdaBoost round, the selected feature is tuned to yield the desired detection rate. In our case, the tuning is done as before by changing the threshold, but it can also be done by removing control points from the feature (if a higher positive detection rate is desired).

The system is down-scaling the image to create a "pyramid" of resolutions, in order to detect pedestrians in different scales. In [Viol 01] Viola and Jones show that this operation can be spared by scaling the feature and not the image. While this can also be done here, this is left for future work.

The system was tested on several test sequences. Its detection rates are similar to the ones obtained without using a cascade. It runs on a regular PC in a speed of 10 images per second. However, from the 100 ms spent on each image, 70 ms are for creating the "pyramid" of images and only 30 ms are for the actual detection. Therefore, if we would have made the learning separately for each resolution, we could avoid the need to create a pyramid and run the system at more than 30 fps. This is better than previously published results.

The cascade contains 16 layers, containing 6, 19, 62, 167, 140, 92, 63, 130, 217, 272, 408, 74, 374, 256, 378 and 157 features respectively.

3.4 The 5x5 moving kernel features

In this section we present another kind of visual feature, to be used with the AdaBoost algorithm. The feature has the following characteristics :

1 It is using differences between sum of pixels in two areas, like the wavelet features. However, instead of just calculating the difference between two rectangular regions, it is applying a 5x5 shape to different locations within a sub-region.

2 It is adopted to the needs of a specialized hardware "smart imaging core" developed in the CAMELLIA project. The European IST-2001-34410 CAMELLIA project, which is

fully described in [Jach 03], focuses on a platform-based development of a smart imaging core to be embedded in smart cameras. This imaging core provides low-level image processing operations like morphological and arithmetic operations, and the high-level applications developed in this context are designed to take advantage of these operations. The features described in this section are designed to be used in the pedestrian detection application of CAMELLIA.

3.4.1 Definition

A *simple classifier* in our context is a 4-tuple $\langle R, E, T, S\rangle$, where $R = \langle R^x_{\leftarrow}, R^x_{\rightarrow}, R^y_{\uparrow}, R^y_{\downarrow}\rangle$ is an image rectangular region, E is a 5x5 linear-filtering element (=matrix) consisting of integers from the set $\{-2,-1,0,1,2\}$, T is a threshold value between 0 and 255, and $S \in \{0,1,2\}$ is the resolution value.

Given a rectangular image region R^{pedes} taken from an input gray image, our goal is to determine if it contains a pedestrian or not. The rectangle must contain the whole pedestrian and be aligned on it with margins of up to 10% from each side. A given simple classifier $f = \langle R^f, E^f, T^f, S^f\rangle$ tests the image region as follows :

- Warp R^{pedes} to target size. Target size is the original size (48x24 in the case of our work with pedestrians) when $S = 0$, quarter size (24x12 in the case of pedestrians) when $S = 1$ and sixteenth of the size (12x6 in the case of pedestrians) when $S = 2$.
- Normalize R^{pedes} to an average pixel value of 128.
- Perform linear filtering, using the element E^f, on the sub-region R^f of R^{pedes}.
- Find the maximum value over the sub-region R^f. If this value is larger than T^f, the classifier's answer is positive. Otherwise, the answer is negative.

A simple classifier of this type attempts to detect a certain shape, designated by its element E, on some region of the rectangle, defined by its region R. Note that a classifier with $S = 0$ detects patterns which take approximately 0.7% of the region surface (like the top of the pedestrian's head), the ones with $S = 1$ detect larger parts (approx. 5% of the region), like the entire structure of the shoulders, and the ones with $S = 2$ detect global structures, as the entire body.

The number of possible simple classifiers of this type is huge (approx. 10^{27}). Intuitively, there is not one single simple classifier of this type that could tell a pedestrian to some high accuracy, but a relatively small set of such classifiers, decided by voting, could definitely accomplish this. The selection of such voting-set instead of using only one simple classifier, is done, as in previous chapters, by the AdaBoost algorithm as in the previous section.

Again, like in the previous chapter, choosing the best simple classifier at each step of AdaBoost cannot be done by testing all the possibilities. Therefore, a genetic-like algorithm is used, which starts with a set of random simple classifiers and iteratively improves them while using the following mutations :

- $\Im^1(\langle R, E, T, S\rangle) = \langle R', E, T, S\rangle$
  where $R' = \langle R^x_{\leftarrow} + \alpha, R^x_{\rightarrow} + \beta, R^y_{\uparrow} + \gamma, R^y_{\downarrow} + \delta\rangle$
  and $\alpha, \beta, \gamma\, \delta \in [-5, 5]$ are chosen randomly
- $\Im^2(\langle R, E, T, S\rangle) = \langle R, E', T, S\rangle$

where E' is produced from E by changing one of its 25 values to a random number in the set $\{-2,-1,0,1,2\}$.

– $\Im^3(\langle R,E,T,S\rangle) = \langle R,E,T',S\rangle$
where $T' = T + \alpha$
and $\alpha \in [-5,5]$ is chosen randomly

The genetic-like algorithm, given in table 3.7, maintains a set of simple classifiers which are initialized as random ones. Each step of the algorithm a new "generation" of simple classifiers is produced by applying the 3 types of mutations on each of the simple classifiers. All 3 mutations are tested and the one with the lowest error might replace the "parent" if it has a lower error. In addition, some random simple classifiers are added at each step.

Although AdaBoost is described in previous sections, it is brought here in table 3.8 to be understood in this context.

3.4.2 Initial experiments

To initially test the features, we ran 10 features training process on a set of 1220 positive and 2072 negative examples of cars. The resulting features are shown in figure 3.24.

Among these features, one can easily observe features that concentrate on the roof, rear lights, wheels, and vertical edges of the car.

3.5 Analysis of features on synthetic data

We have tested the two types of features, control-points features and 5x5-kernel features, to see how well they cope with difficult image conditions. The tests consisted of two types of such conditions : image noise and hiding crosses.

3.5.1 Noise sensitivity

The input data to the process was a 288x384 movie, in which, at first step, the background is totally black and a single white rectangle of size 20x20 is slowly moving in a random manner across the image (see image 3.25). The goal here was to see the functioning of the features is a non-noisy image.

We created artificially 5000 positive example images, each one of size 40x40 pixels, in which a white 20x20 pixels rectangle is posed on a black background. The rectangle was centered on the example images, with a movement of 5 pixels to each direction. We equally prepared 5000 negative examples of size 40x40 totally black. We ran the learning process using the control-points features, and a single-feature AdaBoost was produced. This feature was sufficient to classify 100% of the cases in new video sequences (see figure 3.25). Same results were obtained for the 5x5 features.

Next we added noise, by spreading on each 384x288 image 5000 white dots in random locations. In both types of features, applying the single-feature detector which was prepared in the previous step yielded many false detections. See figure 3.26.

• Input : an error measurement Γ that matches an error to every simple classifier, a number G of generations to run, and a number N of "survivors" at each generation.

• Let Υ be a generator of random simple classifiers. Initialize the first generation's simple classifiers vector C^1 as :
$c_i^1 \leftarrow \Upsilon \qquad i = 1 \ldots N$

• Do for $g = 1, 2, \ldots, G$:

1. Build the next generation vector C^{g+1} as :
   $c_i^{g+1} \leftarrow c_i^g \qquad i = 1 \ldots N$
2. Do for $m = 1, 2, 3$:
   if $\Im^m(c_i^g)$ is valid and $\Gamma(\Im^m(c_i^g)) < \Gamma(c_i^{g+1})$ then
   $c_i^{g+1} \leftarrow \Im^m(c_i^g) \qquad i = 1 \ldots N$
3. Enlarge the vector C^{g+1} with N additional simple classifiers, chosen randomly :
   $c_i^{g+1} \leftarrow \Upsilon \qquad i = N+1, \ldots, 2N$
4. Sort the vector C^{g+1} such that
   $\Gamma(c_1^{g+1}) \leq \Gamma(c_2^{g+1}) \leq \ldots \leq \Gamma(c_{2N}^{g+1})$

• Output the simple classifier c_1^{G+1}.

A simple classifier is *valid* if it consists of a valid image region (i.e. with a positive surface and not exceeding image borders) and of a threshold in the range $[0, 255]$.

TAB. 3.7 – The genetic-like algorithm

• Input : a sequence of N labeled examples $X = \langle (x_1, y_1), \ldots, (x_N, y_N) \rangle$ where $y_i \in \{0, 1\}$
integers T, G specifying number of iterations of AdaBoost and genetic algorithms respectively.

• Initialize the weight vector : $w_i^1 = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negative and positive examples, respectively.

• Do for $t = 1, 2, \ldots, T$

1. Normalize the weights such that $\sum_{j=1}^{N} w_j^t = 1$.

2. Run the genetic-like algorithm G generations, providing it with the error function $\Gamma(h) = \sum_{i=1}^{N} w_i^t \mid h(x_i) - y_i \mid$
The genetic algorithm will return a simple classifier h_t with $\Gamma(h_t) = \epsilon_t$.

3. Set the new weights vector :
$w_{t+1}^i \leftarrow \begin{cases} w_t^i \beta_t & \text{if } x_i \text{ is classified correctly by } h_t \\ w_t^i & \text{otherwise} \end{cases}$
where $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

• Output the classifier :

$$h(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} \alpha_t h_t(x) \geq 1/2 \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

where $\alpha_t = \log \frac{1}{\beta_t}$

TAB. 3.8 – The AdaBoost algorithm, in the same way used by Viola and Jones, with the genetic-like algorithm as a weak learner

FIG. 3.24 – 10 examples of the 5x5 feature, learned for car detection. Each feature is brought in a pair of images (in two columns). On the left of each pair, we have drawn the intensity that this feature responds on that particular image. On the right, we see the 5x5 array of the feature, drawn on the spot where it gives the maximum response for that image.

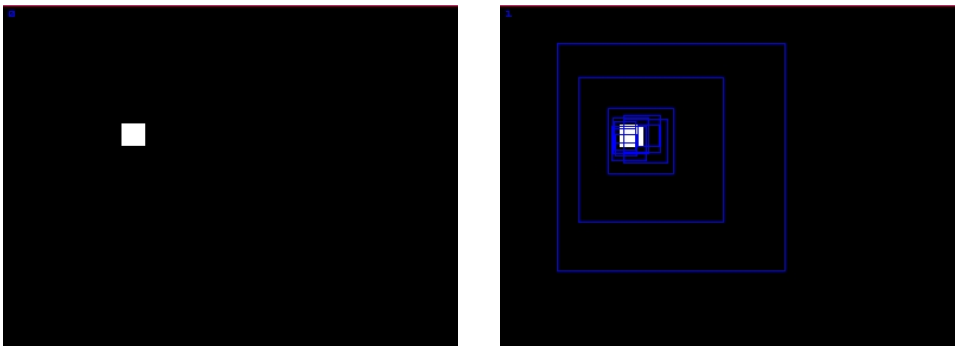


FIG. 3.25 – Left : a non-noisy image with the white rectangle used for the tests. Right : the detection with a single-feature.

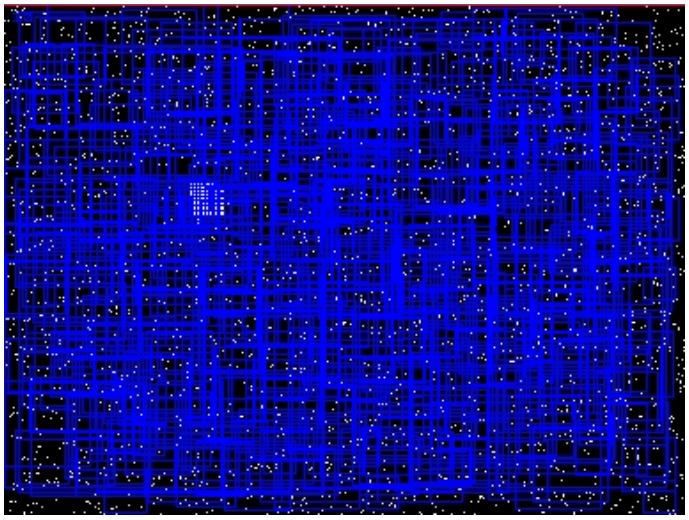


FIG. 3.26 – False detections using a single-feature on the noisy image.

Again, we prepared 5000 positive examples with a white rectangle positioned in the same way as before, but now we added noise made of white points in the same intensiveness as in the noisy film. We also prepared 5000 negative examples with only that noise, without a rectangle. In both types of features we had the same effect : in the learning process, the system could not produce more than one feature (due to complete adaptation to the learning set), but after several "Seville sessions" in which we enriched the negative examples set with harder negative examples, we arrived to a single feature that detected well the white rectangle in all new sequences with 5000 points, as well as with 15000 points per image (see figure 3.27). This feature is shown in figure 3.29, both for 5x5 and control points.

We repeated the learning process with a noise level of 25000 dots per 384x288 image. At this stage, a single control-points feature was sufficient to detect the rectangle, but 5 features were needed to detect perfectly the rectangle with the 5x5 features type. See figure 3.28.

We made the same test with 150000 dots per image. This time we needed 15 control-points features to obtain a perfect detector, as seen in figure 3.30. For the 5x5 type, the same 5-features detector as before was still working. For the control-points, all the 15 features were

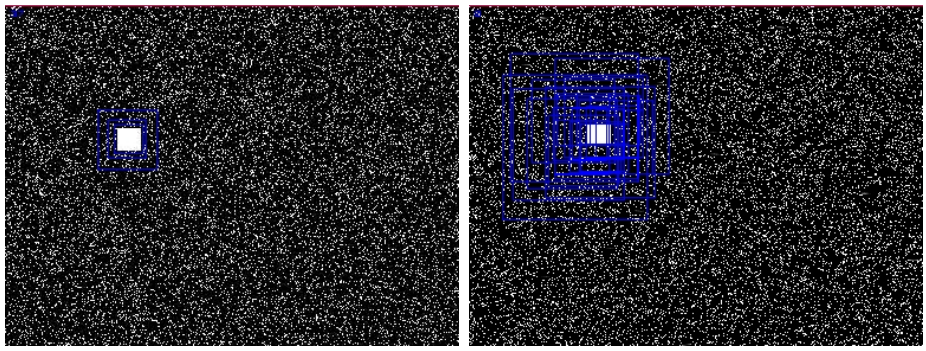


FIG. 3.27 – An image with 15000 random points : the rectangle is perfectly detected by a single feature control-points detector (left) and a single-feature 5x5 feature (right).

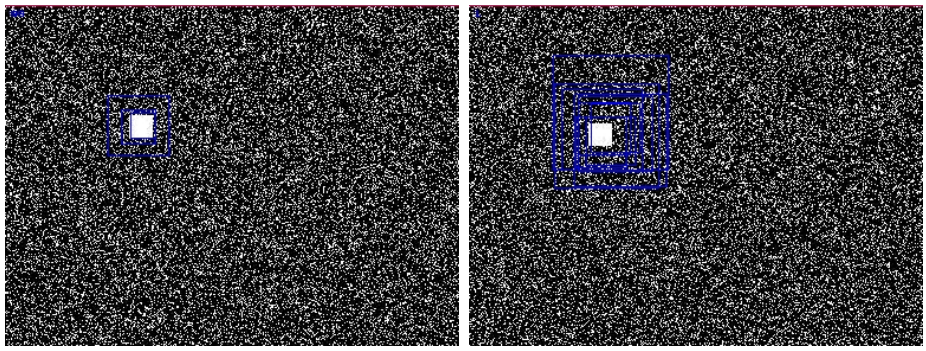


FIG. 3.28 – An image with 25000 random points : we needed a single control-points feature and 5 features of type 5x5 to perfectly detect the rectangle.

of the same type of the feature shown in figure 3.29 : some points inside the rectangle area and some points outside.

In that level of noise, we note that the capability to detect the rectangle is almost as strong as the human capability.

At a noise level of 275000 dots per image, we could not avoid some false detections when using the control-points detector produced in the previous step (with 15 features). The extent was one false detection per two 384x288 images, each image containing 100,000 sub-window examined. As seen in figure 3.31 we see that the false detections are "almost rectangles" - they consist of full white rectangle almost filled. Because these kinds of features cover individual points, they think that these areas are actually rectangles. With the 5x5 type, we managed to produce a perfect detector with 30 features.

A level of 400000 dots per image was already impossible to work with, in both types of features. Also a human could not see if a rectangle is found. See figure 3.32.

To summarize, we concentrated all the numbers of needed features in table 3.9.

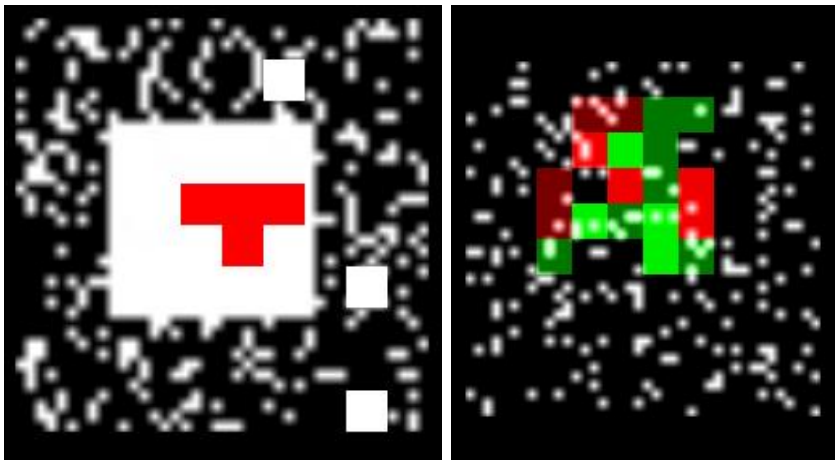


FIG. 3.29 – The single features. The control-points feature compares symmetrically between the inside and the outside of the rectangle (see the section defining this kind of features for a precise definition). Inside, we have 5 points (the dark squares). Outside, 3 points. The 5x5 feature compares the intensity inside the rectangle and at the border.

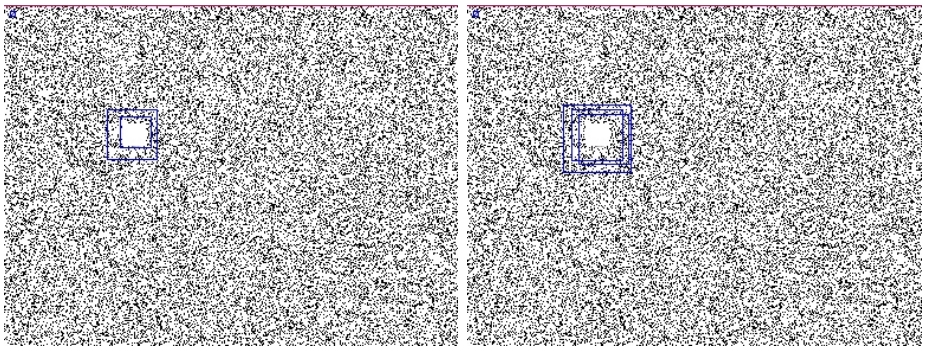


FIG. 3.30 – Tests with heavy noise of 150000 dots per image.

3.5.2 Sensitivity to hiding

Next step we tested how the features cope with hiding of the detected object. We used the same white rectangle, and added 100 cross-like hiding objects in the image. The same process was repeated with 300 such objects. In both cases, we managed to produce a 30-features control-points detector that detect perfectly the white rectangle without false detections (see image 3.33). A 30-feature 5x5 detector was also created, and also detected perfectly the two cases.

Later we increased the number of hiding objects to 1500. At this point, all the rectangles were detected by the control points, but also were detected some false positives where black areas were formed in the absence of hiding objects (see image 3.34). This is explained by the fact that the control points features are symmetric. The 5x5 detector continued to perform perfectly on this problem.

When using 3000 hiding objects, no detection was possible (see image 3.35). However, even a human could not reliably spot the rectangle.

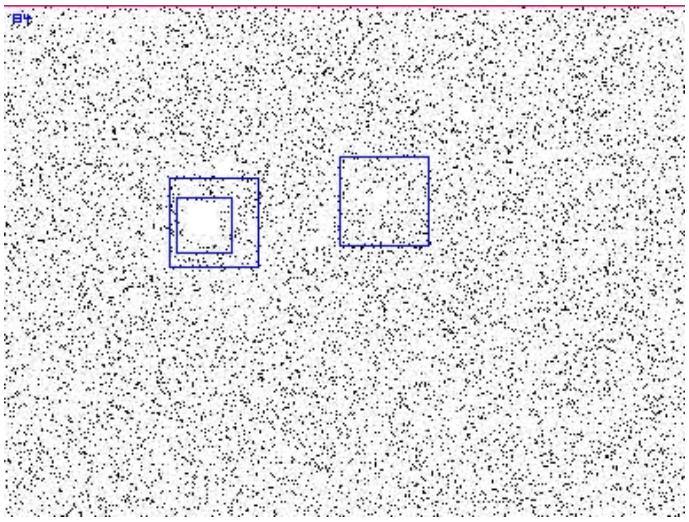


FIG. 3.31 – False detections in a noise level of 275000 dots per image, with a 15-features detector.

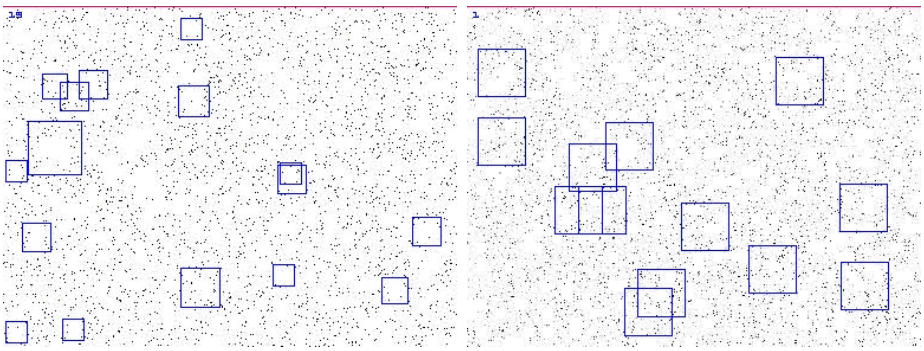


FIG. 3.32 – False detections in a noise level of 400000 dots per image, with both feature types. Even a human eye cannot reliably find the rectangle here.

The sensitivity of both features to hiding objects is summarized in table 3.10.

3.5.3 Conclusions

The 5x5 features are obviously more stable for noise, because they test regions and use thresholds. In the noise, the control-points feature needed more and more points to "cover" the problem each time, whereas the number of features needed for the 5x5 features was rather stable.

With hidden objects, the strength of the 5x5 is also shown, but less than in noise. The reason is, perhaps, that the hiding objects are not really "noise" ; there is no need for thresholds, but more to detect shapes. In detecting shapes, the control-points features are getting close to the capability of the 5x5 features.

TAB. 3.9 – Number of features needed for perfect detection of the white rectangle as a function of the noise intensity.

Number of random points per 288x384 image	Number of control-points features needed	Number of 5x5 features needed
0	1	1
5000	1	1
15000	1	1
25000	1	5
75000	7	5
150000	15	5
275000	-	30
400000	-	-

TAB. 3.10 – Number of features needed for perfect detection of the white rectangle as a function of the hiding objects intensity.

Number of hiding crosses per 288x384 image	Number of control-points features needed	Number of 5x5 features needed	remarks
0	1	1	
100	30	30	
300	30	30	
1500	30	30	Control points failed symmetrically
3000	-	-	

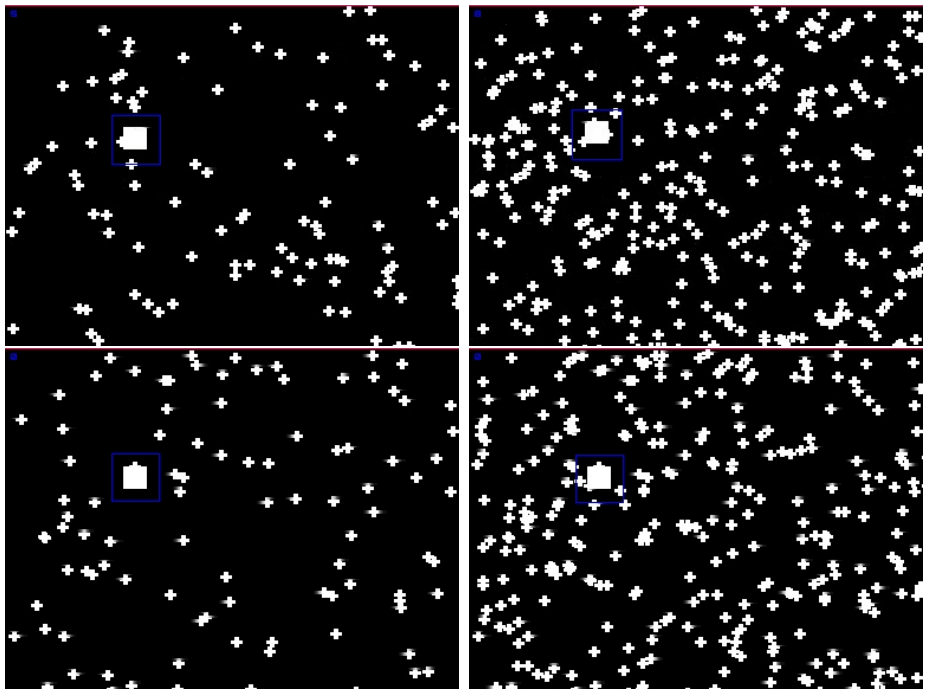


FIG. 3.33 – White rectangle among 100 and 300 hiding objects, detected by a 30-feature detector. Top : control points feature. Bottom : 5x5 features.

Control-points features : statistical explanation

A good understanding of how control-points features work with noise can be obtain by observing the following :

- a 40x40 rectangle contains 1600 pixels. From these, 20x20 (400 pixels) are filled with the white rectangle. The remaining 1200 contain the noise.
- Assuming the noise intensity is such, that $0 \leq p \leq 1$ of the 1200 pixels described above are white.
- Assume all 20x20 white rectangles are in the middle of the 40x40 detection window, and that the average feature is testing 3 pixels inside the rectangle region ("the outside control points") and 3 pixels outside ("the inside control points"). Such a feature is shown in figure 3.29.
- The probability of a single outside control point to fall on a noisy (white) pixel is p. The probability of all three outside points to avoid a noisy pixel is $(1-p)^3$.
- For a feature to succeed on a positive example (i.e. classify positively) the above is enough. So the probability of a single feature to say "yes" on a positive example is $P_p = (1-p)^3$. The "no" probability is $N_p = 1 - P_p$.
- To fail on a negative example (i.e. classify positively) a feature has to fall on white pixels on all 3 inside control points, and on black pixels outside. The probability for that is $P_n = p^3 * (1-p)^3$. So the success probability is $N_n = 1 - p^3 * (1-p)^3$.
- Assume we set the boosting threshold on $0 \leq t \leq 1$. Assume we use N features. For a positive example, we need more than $N * t$ features to say "no" in order to fail. This has

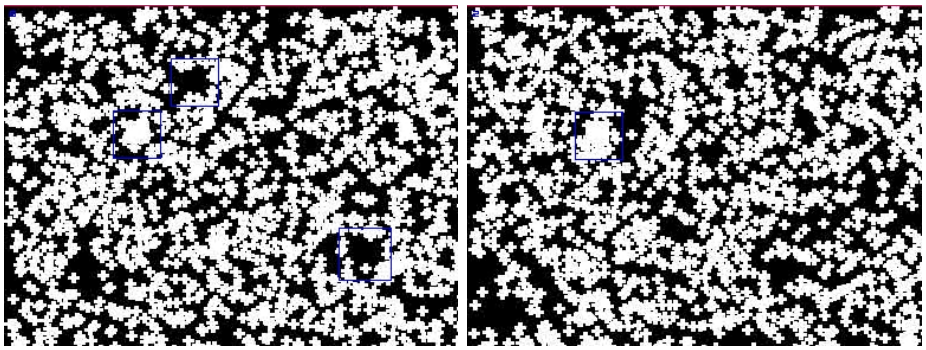


FIG. 3.34 – White rectangle among 1500 hiding objects, detected by a 30-feature detector. Left : control-points feature with some false detections. Right : 5x5 features.

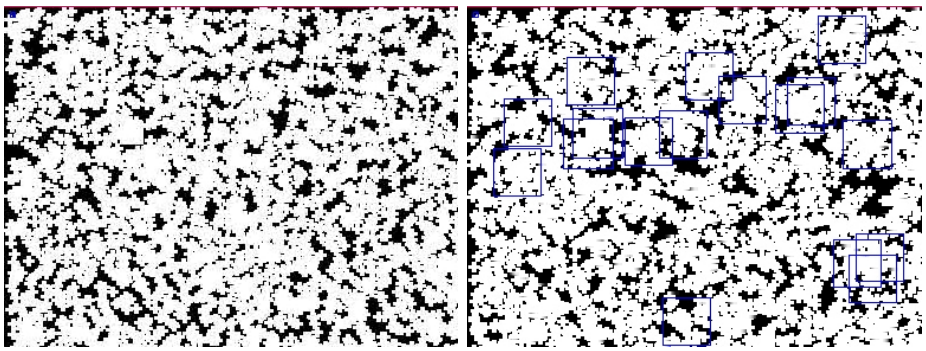


FIG. 3.35 – White rectangle among 3000 hiding objects : no detection was possible. Left : control-points. Right : 5x5.

a probability of N_p^{N*t}. For a negative example, we need more than $N*(1-t)$ features to say "yes" in order to fail. This has a probability of $P_n^{N*(1-t)}$.

– If we use $N = 10$ and $t = 0.5$, then values of 5000,15000,25000,75000, 150000, 275000 and 400000 white points per 288x384 image correspond approx. to p=0.05, 0.1, 0.2, 0.35, 0.6, 0.75, 0.9 respectively. The corresponding probabilities of error on positive example are 0.000059, 0.001, 0.027, 0.2, 0.71, 0.92 and 0.99. On negative examples these are $1.41*10^{-20}$, $2.05*10^{-16}$, $1.15*10^{-12}$, $2.2*10^{-10}$, $5*10^{-10}$, $1.2*10^{-11}$ and $2*10^{-16}$.

Observing these numbers, we see that until 75000 we have a good detection rate with false detection rate which is very close to zero. On 150000, however, we start to have low detection (0.71 failure means 29% detection). To handle this, we can add features (say $N = 25$) and elevate the threshold (say $t = 0.9$). Then we have 95% detection rate with 5e-10 false detection rate, which is good.

Until now, the numbers conform with the results of table 3.9. However, when we move to 275000 points, we expect to find out that no detector is possible. According to the theoretical calculations, this is not correct. If we choose $N = 200$, $t = 0.97$ we get 95% detection rate with $8*10^{-14}$ false detection. In fact, it turns out that for every amount of noise (which is not 100%) we can theoretically reach high detection levels. Imagine a situation where 99%

of the points in an image are noisy (i.e. white). Then, $p = 0.99$. If we choose $N = 2500000$ and $t = 0.999999$, we get 91.5% detection rate and $9 * 10^{-16}$ false positives.

This is, of course, not true in the practical case, where image is made of individual pixels ; because in many 40x40 rectangles there will not be 99% noise but 100% noise (i.e. the rectangle will be all white), in which case no success is possible. If the image was continuous, the conclusion above would have been true.

3.6 Analysis of features on real data

We have tested the different kinds of features on real video sequences taken from an on-board frontal camera. The sequences were taken in the Parisian area in France at different times of day and lighting conditions.

We marked by hand 323 images from these video sequences. In each image, all the locations of cars were exactly marked. Then, we applied the various detectors on these images, compared the detections to the ground truth and created ROC curves, using the methods described in section 3.1.

3.6.1 Without normalization

To understand the real quality of the various features, one has to consider the time domain. Therefore, we have generated a 3D curve showing the various 2D ROC curves that can be obtained in relation to the detection time. We compared three kinds of features :

- The control-points features of the kind described in section 3.3
- The 5x5 features described in section 3.4
- The rectangular features used by Viola and Jones [Viol 01].

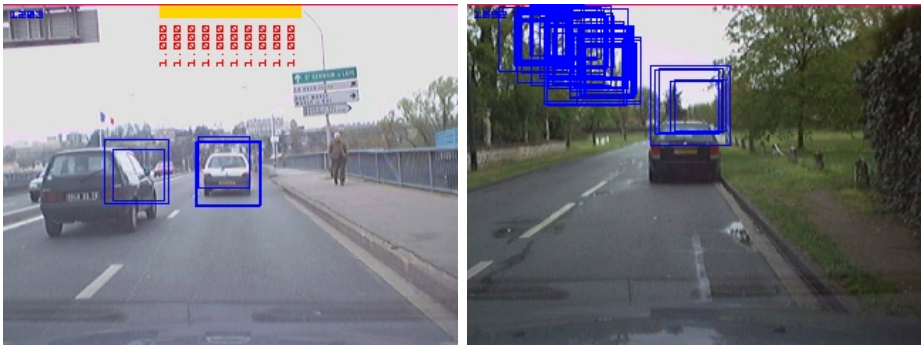


FIG. 3.36 – The effect of illumination on Viola and Jones features (without prior normalization of the image). On the left, we see how the features function quite well when good illumination is on the scene (for the bright car). On the right, we see that once the scene is dark, the feature detects false positives where light is found.

The 3D ROC graphs are shown in figure 3.37. In these tests, we did not normalize the input images before applying the features. One can see that this fact deteriorates the results of the Viola & Jones and the 5x5 features, because they rely on thresholds. In fact, in

these 3D ROC curves, the control-points features are far better even without considering the performance issue. To better understand this, we bring in figure 3.36 a demonstration of how VJ features depend on the illumination.

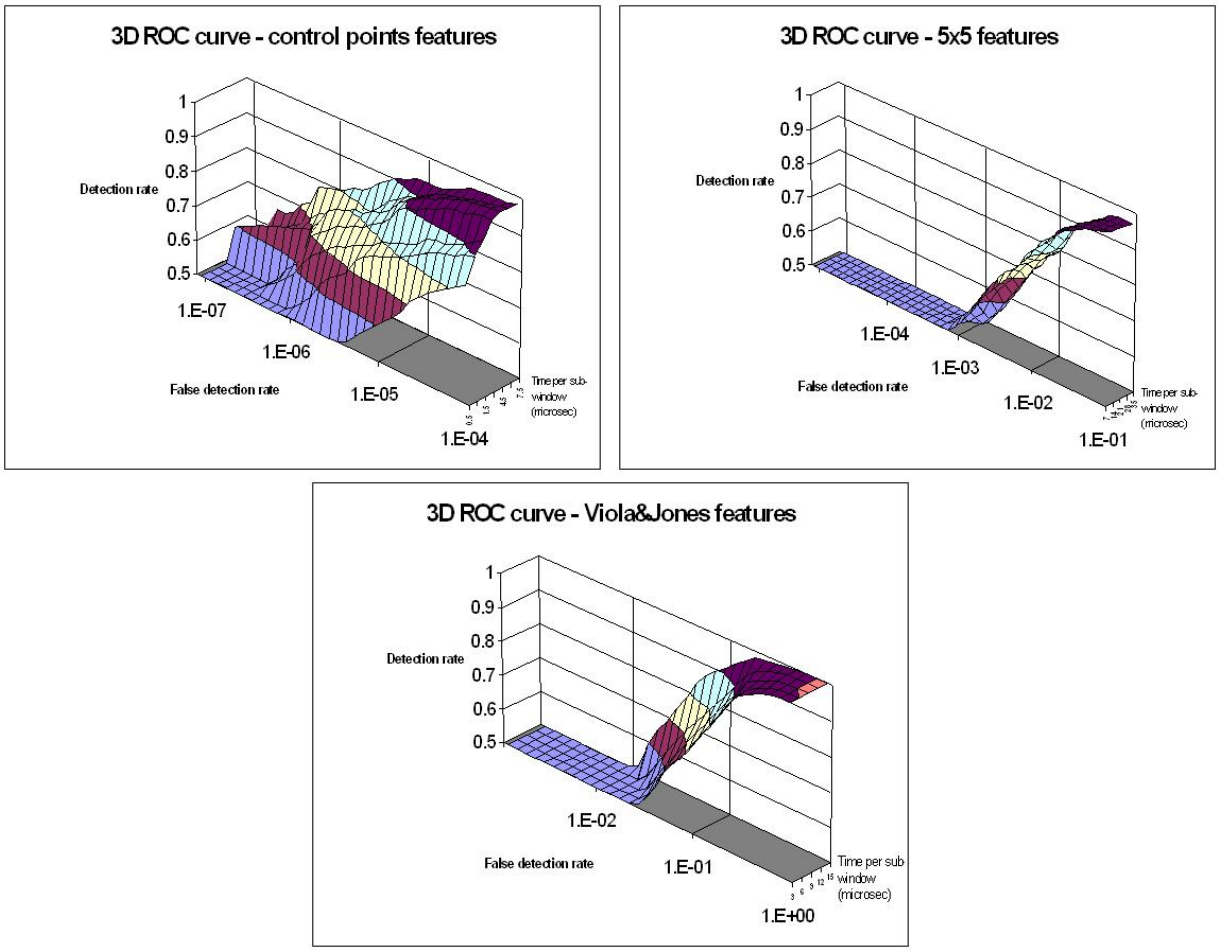


FIG. 3.37 – The 3D ROC graph for various kinds of features. The X and Y axes are of the same kind of the conventional (2D) ROC curve explained in the text previously. The Z axes is the time in microseconds needed to apply the detector on one sub-window. Obviously, the control-points features are better in this case.

3.6.2 With normalization

To better evaluate the 5x5 and Viola/Jones features, we normalized each examined sub-windows prior to applying these features. The results change, such that the VJ features are better, when given enough time to operate (about 15 microsecond per sub-window). The control-points features rest better, when time is limited (about 6 microsecond per sub-window).

3.7 SEVILLE : SEmi-automatic VIsuaL LEarning

The use of machine learning methods for visual object detection has become popular in recent years [Viol 02, Papa 98a]. It is now generally accepted that detection algorithms developed using machine learning are more accurate, more robust and faster than hand-crafted ones.

However, for these methods to work well, one needs large amounts of labeled training data. It is becoming increasingly evident that the manual work involved in hand labeling images has become a major factor in the time it takes to develop an accurate object detector. Levin et al. [Levi 03] used co-training in the context of visual car detection to improve the accuracy of a classifier using unlabeled examples. In this document we use active learning in the context of visual pedestrian detection to concentrate the human feedback on the most informative and hard to classify examples.

An important observation in this context is that while it is relatively easy for a human to identify pedestrians in images, it is very hard to identify those elements of the image that are not pedestrians but are "hard" to identify as such. Clearly, this depends heavily on the details of the detection method and changes learning progresses. It thus makes common sense to select for labeling those examples that the learning system, in its current state, finds hard to classify or "borderline".

While this makes sense on an intuitive level, it is hard to justify it when one is working with a *generative* model for images as it creates a heavy bias in the data sampling process. It is easy and natural to justify selective sampling when the goal is to learn a classification function, rather than a distribution.

As a realization of our method, we present the details of a system named SEVILLE (SEmi automatic VIsuaL LEarning) [seville] which is running AdaBoost [Freu 95], in the same way used by Viola and Jones [Viol 02] but with a different set of features.

3.7.1 The setup

We used quarter PAL (384x288 pixels) color video sequences that were taken from a forward-looking camera mounted on a car. The video was recorded as the car was driven on variety of rural and urban roads in and around Paris. The overall length of the recorded video was about 6 hours, from which we have cut short sequences of duration ranging from several seconds to several minutes each. It will be noted that these sequences were cut from the parts containing pedestrians, which is about 30% of the entire data.

The color images in our data were transformed to grey-level images in order to reduce computation. We use images 48 pixels high and 24 pixels wide. We define an image to be a true positive detection of a pedestrian if it contains the complete body (possibly partially occluded) with margins of approximately 20% of the body size around it.

The video data contains approximately 250 different pedestrians per hour. When using these pedestrians for training, we usually took one image out of 3-4, in order to avoid examples that are too similar to each other.

In the images that contain pedestrians, usually 1-4 appear at the same time. Approximately third of the pedestrians are walking on sidewalks, parallel to our car, where the rest

are crossing the road, mostly on crosswalks. It will be noted that crossing pedestrians are often using various diagonal crossing directions, so our database covers all viewing angles fairly well. The scenes contain various lighting conditions and various types of complex background.

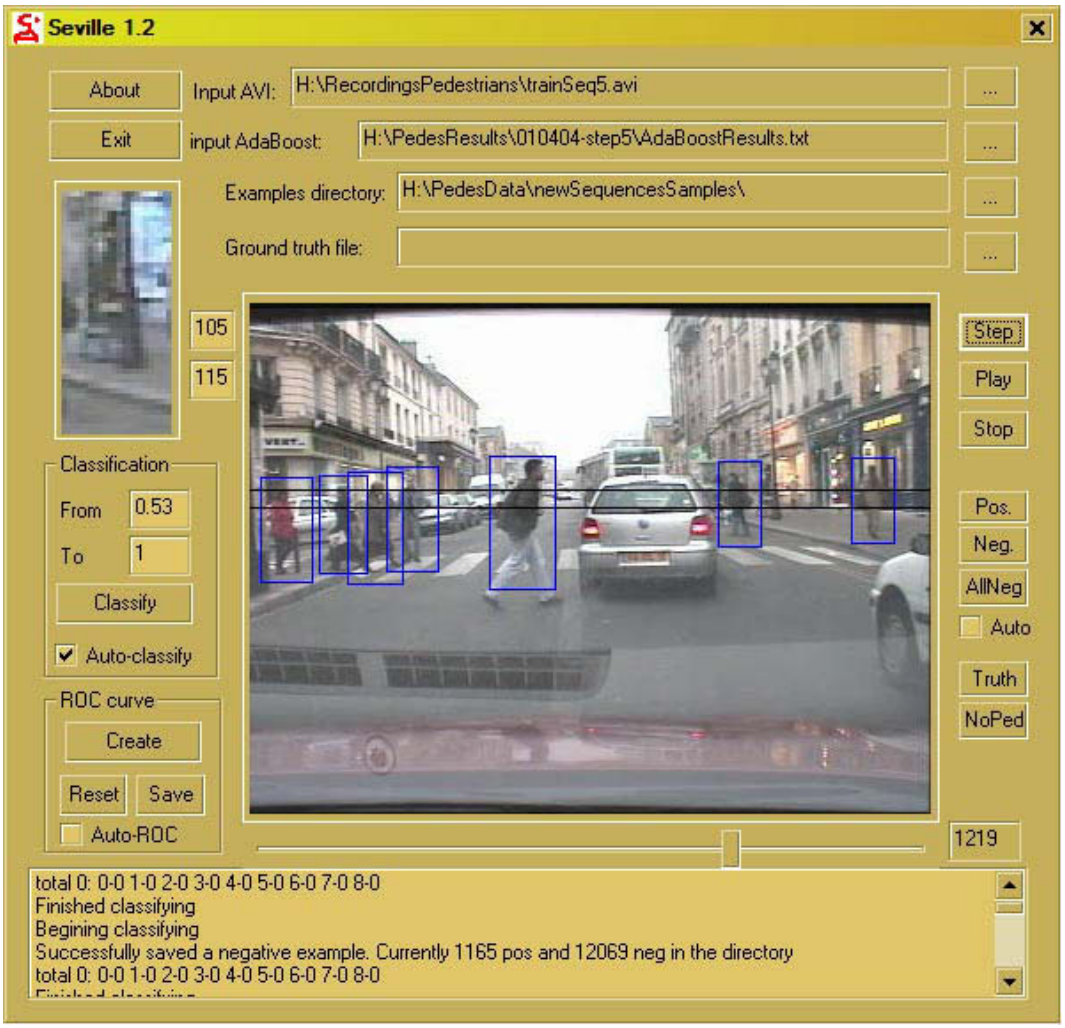


FIG. 3.38 – The Seville user interface.

The visual features

Instead of the wavelet-based weak classifier used in [Viol 02] we used the illumination-independent features described in section 3.3, because they run faster and are easier to implement. Wavelet features are calculating the difference between the sum of the pixel values in some rectangular regions (see figure 3.39), and comparing this difference to some fixed threshold. While these features capture well the pedestrian class, they depend on the luminosity of the image and demand histogram equalization for each examined sub-window. Illumination independent features were shown to yield the same detection results while consuming less image access operations and without resorting to histogram normalization. These features

work in 3 resolutions (full, quarter and sixteenth) and compare the pixel values in two sets of "control points" in the image. The feature classifies positively if and only if the values of *all* the points in one set is superior of the values of *all* the points in the other set. The feature does not use any threshold and can detect objects independently of the distribution of the image histogram. Applying this feature on a sub-window demands approximately 4 image access operations.

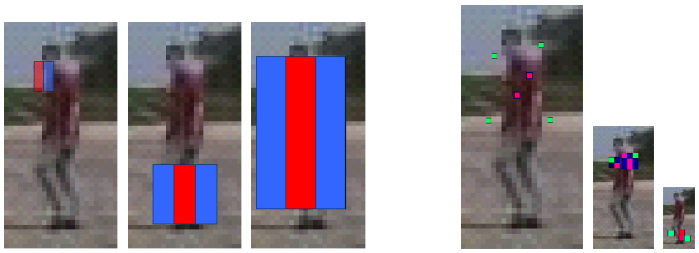


FIG. 3.39 – Examples of wavelet features used by Viola and Jones (left) and the illumination independent features used in this work (right). Wavelet features are calculating the difference between the sums of the pixel values in the red and the blue region, and classifying positively when this difference is superior of a fixed threshold. Illumination independent features work on 3 resolutions (full, quarter and sixteenth) and examine the image in single-pixel "control points". The feature is classifying positively when all the pixel values in the red points are larger than the pixel values in all the blue points

The learning algorithm

The learning algorithm we used is AdaBoost based on single-feature decision stumps. We use the same learning procedure as Viola and Jones [Viol 02], only our choice of features is different. We assume that each visual feature f_i outputs +1 or -1 and the polarity is set such that each feature is positively correlated with the detection event, so that the feature weights are all positive. We *normalize* the total score generated by AdaBoost using the sum of the feature weights, $s = \sum_i \alpha_i f_i / \sum_i \alpha_i$, so that the total score always ranges between -1 and +1.

Since it is too computationally expensive to check all possibilities of visual features in each AdaBoost step, we used a heuristic search based on genetic programming. Our search procedure starts with a first "generation" of 100 random features, and uses various types of crossovers and mutations in order to improve the features to yield minimal weighted error on the training set. Our search stops when no improvement is obtained during 40 consecutive generations.

The interactive labeling system

Our experiments were carried out using a software system named SEVILLE which provides a graphical user interface for interactive labeling of training examples. The system is playing an input video sequence while possibly applying an existing AdaBoost classifier on a set of sub-windows in the image, in different scales and locations. The examined sub-windows are starting in the standard size (24x48) and scaling up 10% each time. To reduce

computations, we defined a horizontal line - the scene horizon - and used only rectangles which intersect this line. Altogether, the classifier examines around 170,000 rectangles for each image.

The user can instruct the system to display all sub-windows of which classifying score falls inside a specified range. These rectangles can be selected (individually or by groups) and be turned into learning examples by labeling them as negative or positive. Such rectangles are warped to the predefined standard size (24x48) and put as JPEG images in the appropriate directory, according to their label. The system allows the same treatment for rectangles which are marked manually (i.e. by dragging the mouse).

The idea is that in mature stages of the examples collection process, one can use the already-collected examples to build a good classifier, which in turn can aid in the selection of examples. Such classifier can point out which examples are hard to separate and the role of the human labor is only in telling "positive" or "negative" for each of these examples. This task eventually becomes much faster than manual collection of labeled examples.

3.7.2 The design of the experiment

Most machine learning procedures consist of four steps : data collection, labeling, training and testing. As we discussed earlier, in visual detection problems, labeling is a very labor intensive step. The goal of our experiment is to demonstrate how an iterative procedure which alternates between training and labeling can be used to substantially reduce the work involved in labeling.

In order to ensure an unbiased measure of the test accuracy of our detector we started by collecting 215 pedestrians and 37,064,791 non-pedestrian images, chosen from a video sequence of 2 :08 minutes. We later use this data-set to measure the ROC of the detectors generated by our process. Note that in order for the system to be effective we need the false positive rate to be around 10^{-5}.

Each step in our procedure consists of two phases : collection of labeled examples and training. The training phase uses *all* of the labeled examples collected so far, separates them into 2/3 for training and 1/3 for validation and then runs the boosting algorithm on the training set until the ROC curve on the validation set stops improving.

Each labeling phase (other than the first) uses the detection scoring function generated by the training phase of the previous step. Each labeling phase uses a new, previously-unused video sequence. When applying the detector, the system is instructed to show detections whose boosting score (a number ranging between -1 and 1) is higher than μ^- and lower than μ^+. These values are chosen manually, such that approximately 0.1% of the negative examples and around a half of the positive examples are shown.

Once detections appear on the screen, the GUI is used to label each detection and to turn it into a training example. Some detections can remain unlabeled if is not clear if they are positive or negative. All the labeled examples are added to the training set and the following training phase is executed.

3.7.3 Theoretical justification

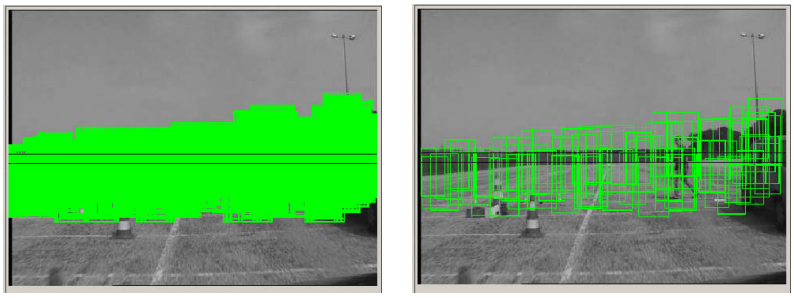
The theoretical justification of considering the normalized boosting score $s = \sum_i \alpha_i f_i / \sum_i \alpha_i$ as a measure of prediction confidence was provided by Schapire et al in [Scha 98]. Intuitively, the normalized score value of a random test example varies only little if we select different training sets *from the same distribution*.

However, in our application we take one step further and assume that we get an accurate classifier even if we significantly alter the distribution of the examples by selecting mostly those examples that are close to the decision boundary. Note that if we were creating a generative model of the data, then such skewing of the distribution would be likely to introduce significant bias into our classifier. However, our goal is to minimize classification error, not to maximize model likelihood, which is the reason that our learning algorithm can tolerate such skewing of the distribution of examples.

This is not true in general, but is true when the initial unbiased sample is sufficient to restrict the set of good classifiers to a unique approximation of the optimal classifier. In other words, the distribution of the scores generated by AdaBoost is such that all convex combinations of the weak rules that are significantly different from the one we found have error that is much higher than the one we found. As a result, we can eliminate these classifiers from consideration and concentrate on identifying the optimal classifier that is close to the one we have already found, which means that we only need to know the labels of examples that are close to the decision boundary.

As we shall see in the experimental results. This condition indeed holds in our case and so the procedure is justified.

Of course, the ultimate proof of the effectiveness of our method is given provided by considering the error of the final classifier on held-out test data.



(a) (b)

FIG. 3.40 – Use of the initial classifier in step 2 - (a) original detections (b) with grouping.

Step	positive examples	negative examples	Weak rules	μ^-	μ^+	data length	human labor	training time
1	6	10	1	-1	1	0 :06	3m	2s
2	36	374	3	-1	1	0 :14	3m	8s
3	46	520	7	0.6	1	0 :24	10m	30s
4	86	1332	30	0.4	1	1 :05	20m	2m
5	182	2675	59	0.1	0.8	1 :36	30m	10m
6	417	7804	270	0	0.6	3 :03	2h	1h20m
7	848	8236	893	-0.02	0.5	3 :31	5h	9h
8	1178	16613	1500	-0.02	0.5	5 :14	8h	28h

TAB. 3.11 – The details of the training steps. Data length is in format MM :SS. Amount of human labor and training time is cumulative (h=hours, m=minutes, s=seconds).

Step	positive examples	negative examples	weak rules	human labor
1	15	57	1	3m
2	52	215	12	10m
3	73	527	18	20m
4	98	832	40	30m
5	291	2207	82	2h
6	1012	7658	330	5h
7	1183	13175	2250	8h

TAB. 3.12 – The details of the training steps, second time.

FIG. 3.41 – The simple and complex sequences used in the initial and mature steps, respectively.

FIG. 3.42 – Results of the classifier in step 8 on two independent test sequences.

3.7.4 The experiment

The starting step

The process started by using a simple sequence (see figure 3.41) taken on a parking space, to collect 15 positive examples (pedestrians) and about 57 negative ones (non-pedestrians) by hand. This is done by playing the video sequence, marking rectangles over pedestrians and non-pedestrians, and telling the software to transfer these rectangles to the positive or negative examples directory, using buttons "NEG" and "POS" (see figure 3.38. The system automatically warps the rectangles into the standard size (24x48) and stores the examples as uncompressed JPEG images.

Then, one round of AdaBoost was performed, obtaining one weak-classifier which was already overfitting because it had 100% detection rate with zero false positives on this tiny training set.

The second step

The classifier obtained at the first step is applied to consecutively classify all the sub-windows in some images in the same video sequence as well as three other simple "parking" sequences. As mentioned above, in our configuration of 24x48 pedestrians, the system examines about 400,000 sub-windows of each image, consisting on all the possible sub-windows in 16 scales, starting with the basic size of 24x48 and enlarging the size in 10 percent each time (26x53,29x58 etc). These parameters can be controlled by the software.

FIG. 3.43 – Positive (left) and negative (right) examples for the different steps (each line is a step, from top to bottom). These examples received the minimal (for positive examples) or maximal (for negative examples) score during classification in their respective step.

Those sub-windows which were positively classified by the classifier are marked with a dark rectangle. Obviously, there were many such rectangles and it is almost impossible to extract any useful information at this step (see figure 3.40. A special option in the software groups adjacent rectangles and makes the situation a little bit clearer.

We now continued to collect more examples. The collection of negative examples became much easier now, as each of the dark rectangles (the grouped "detections" of the classifier) could be selected by the mouse and turned into a negative example using the button "NEG". Moreover, when there was a part of the sequence which was verified not to contain correct detections, we could use a special button "ALL NEG" which transfers *all* the detections into the negative examples repository. Using this function we could obtain many negative examples in a few moments, enlarging the set to 215 negative examples. One can see in figure 3.43 that these negative examples are already better than random negative examples, since they do not contain "trivial" negative examples like the sky.

We used the same method to collect positive examples. However, at this step it was not always possible to do it because many times the pedestrian was not detected, so we also marked some pedestrians by hand (by dragging the mouse) as in the first step. The total

FIG. 3.44 – Three positive images (left of each five) and two negative images (right of each five) for the seven steps done in the second time (each five images is a step, from left to right, top to bottom).

number of positive examples was now 52.

We now divided the examples set to training set ($\frac{2}{3}$ of the images) and validation set (the remaining $\frac{1}{3}$), and ran 25 cycles of AdaBoost. Testing the resulting strong classifier on the validation set showed that 10-12 weak classifiers were enough at this stage, so we used only 12 weak classifiers in the strong classifier used for the third step.

Further startup steps

The next three steps, as the first two ones described above, can be characterized as startup steps because the amount of data is still not large. Due to this fact, training graphs are still not fully reliable and intuition plays an important role in selecting the various parameters. Also, in these steps, positive examples are many times collected by hand (that is, by marking a rectangle over a pedestrian) rather than using detections.

At each of the steps 3,4 and 5, the strong classifier produced in the previous step was again used on a *new* video sequence. This time we started to use urban sequences (see figure 3.41). Now, the program is directed to present only sub-windows whose boosting value (the sum of the weights of all the weak classifiers who classified positively) was larger than a value μ^-. This value was calculated such that 99% of the negative examples in the validation set received less than μ^- from this classifier.

We then selected those rectangles which are non-pedestrians and turned them into negative examples. Once again, by using only these "hard" negative examples we ensured the high quality of the examples set.

We also selected those detected rectangle which are pedestrians (the correct detections) and turned them into positive examples. Note that in this case, we do not limit ourselves to "hard" positive examples but take all of them, because of the need to arrive to a large number of positive examples in these startup steps. In later, more "mature" steps, we could activate an option in the software, causing the system to actually put into the positive examples set *only rectangles which received less then a value* μ^+. This value is calculated such that 90% of the positive examples in the validation set received more than μ^+ from this classifier.

In the case that a pedestrian was not marked by the system (that is, it received a boosting

value of less than μ^-), we mark it "manually" (that is, by dragging the mouse) and add it to the examples set as a positive example.

The thresholds of 99% and 90% were chosen arbitrarily ; one could choose different thresholds (provided that the validation set is large enough) and obtain negative and positive examples of a higher quality less rapidly, or the other way around. In our experiments, we started with these values, then changed the negative threshold from 99% to 99.9% when the validation set was large enough.

At each of the steps 3,4 and 5 we enlarged the number of positive examples to 73,98 and 291 and the number of negative examples to 527, 832 and 2207, respectively. After each step, we followed the procedure described in step 2 and divided the examples set to training ($\frac{2}{3}$) and validation ($\frac{1}{3}$). Then we ran AdaBoost and decided about the number of weak rules according to the results on the validation set. In these 3 steps we used 18, 40 and 82 weak rules.

The mature steps

In the following steps we followed a strict procedure of examples collection, training and validation. At each of the steps, as in previous ones, we used the classifier trained in the previous step to classify all the sub windows in new video sequences. However, while the startup steps we used one short video sequence of 3-4 minutes containing 2-3 different pedestrians each time, we now used at each step about 15-30 minutes of video containing dozens and later hundreds of different pedestrians. Each time a classifier was used, we tuned the software to put into the positive examples repository only those who received a boosting value of less than μ^+ and negative ones which received more than μ^-. As mentioned above, at some stage we leveraged μ^- a little bit so that 99.9% and not 99% of the negative examples received a boosting value smaller than μ^-.

At the time of writing this document, we executed two mature steps, enlarging the examples set to 1012 and 1183 positive examples and 7658 and 13175 negative examples, respectively. The number of weak rules used in the strong AdaBoost classifier is 330 and 2250, respectively.

Figure 3.45 shows the ROC curve of our classifiers in the different steps, tested on 3 video sequences which did not take part in the learning process. We see that throughout the steps, results are improving. Note that the problem in question is pedestrian detection in *any angle* (not just forward looking pedestrians), in a relatively small resolutions (24x48 pixels), using *gray images* and without using any motion information. Figure 3.42 shows examples of successful detection in complex urban environment, using the classifier obtained in the last training step. Some movies can be found in the SEVILLE website.

3.7.5 Experimental results

We ran eight steps of labeling and training. The growth of the examples set in the different steps is shown in table 3.7.3. In the table, we show also the number of AdaBoost cycles (weak rules) used in each step, the values μ^- and μ^+ used in the collection process, the length of the input sequence and an approximation of the human labor and the training time that

was needed to complete that step. (Training was done on a standard PC with a single 3GHz Pentium processor.)

Much insight into the active learning process can be gained from looking at the type of mistakes that are dominant after each step. In figure 3.43 we show the positive examples with smallest scores and the negative examples with largest scores as a function of the step number. During our experiments so far, we identify three types of steps :

Steps 1-3 can be considered as "startup steps", where no valuable detector is available, positive examples are collected mostly manually, while negative examples are collected from the numerous false detections (see figure 3.40). As mentioned above, in these steps we used simple and short sequences (see figure 3.41), where the goal is only to obtain a stable detector. The mistakes made during the startup steps are trivial.

Steps 4-6 can be described as intermediate steps, where a valuable detector exists and where quality of the examples produced is already much higher than that of examples collected randomly by hand. However, these steps are not yet "mature" in the sense that trivial negative examples are still appearing and pedestrians in the new sequences are not always detected. The mistakes that are made during these steps are on image frames that contain complex backgrounds.

Steps 7-8 are mature steps, where μ^- and μ^+ are stabilizing around -0.02 and 0.5 respectively, examples are hard to separate and most of the new pedestrians are detected. The mistakes made by the detector on these steps are much more subtle. Some of the false positive examples in steps 6-8 are mistakes that might be made by a human (using this restricted view and no movement or color information). Some of these look pedestrians, and others are actual pedestrians but the image is not well positioned in the detection frame.

The experiment was conducted once more. The results of the second time are presented in table 3.12 and figure 3.44, in the same format described above.

Quantitative results

We tested the detectors obtained by the different steps on the test data. The resulting ROC curve is shown in figure 3.45. One can observe that the detector improves as the process advances. Some successful detections from the test data are shown in figure 3.42. We can see that significant gains are made in the false positive rates throughout the process.

In figure 3.47, the ROC curves for the validation set show that boosting makes continual progress throughout the 1895 cycles of the 8th step. Note that the ultimate validation error is much worse than the actual test error because the validation set, like the training set, is biased towards hard examples. Also note that the ROC curve continues to improve much beyond the point at which the separation between positive and negative examples on the training set is perfect. A phenomenon that is explained by the increase of the separation margin between the positive and negative training sets (indicated by the bold red and blue curves on the bottom right graph).

In figure 3.48 it is interesting to analyze the difference between the distribution of scores on the new labeled examples that are added at iteration 8 with the examples that we added in earlier steps. One can clearly see that the main difference is that the new negative examples have significantly higher scores, or, in other words, they are the ones that are harder to

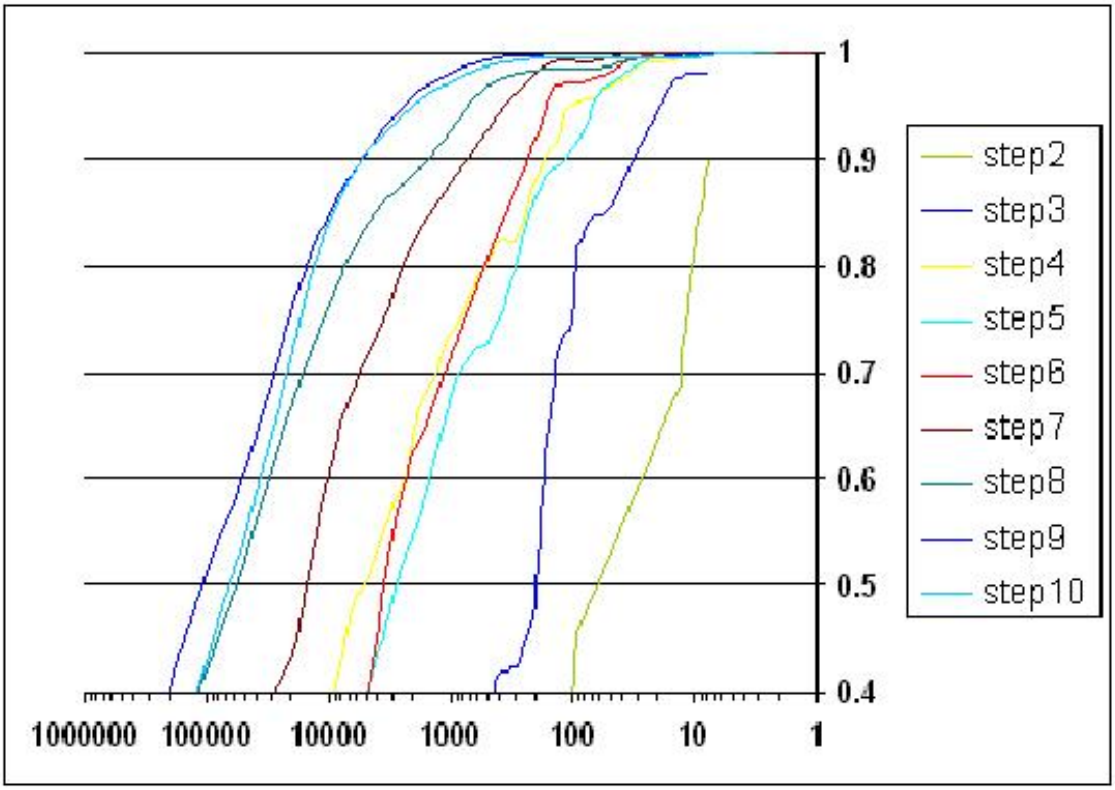


FIG. 3.45 – The test-set ROC of the detectors generated in each of the 10 steps.

separate from the positive examples.

Figures 3.49 and 3.50 show the training graphs for steps 3-8 (this data is quite meaningless for the startup steps 1-2, and to some extent even to step 3, as seen in the figure). In the graphs, we can see the positive and negative error as a function of the number of weak learner. We see that each step we need more weak learners to reach approximately the same rates. Nevertheless these rates are reached each step, and this is a good sign : even as we add more and more difficult examples, the rates remain the same.

3.7.6 Observation at the separation process

It is interesting to observe the learning process as it separates the positive and negative examples. In figure 3.51 we see the histograms of the boosting values of the examples with different numbers of weak learners. The green data are the positive examples, the red are the negative ones and the blue are unlabeled examples chosen randomly from the input images. Naturally, the large majority of these examples are negative examples, but not hard-to-separate as the labeled negative examples. This is why they get a slightly smaller boosting value, as one can see in the histograms.

We see that with 1895 weak learners, the validation set is almost totally separated - one can select a threshold above which almost all are positive and below which almost all are negative. We said "almost" - because there are still outliers which are not separable. One can

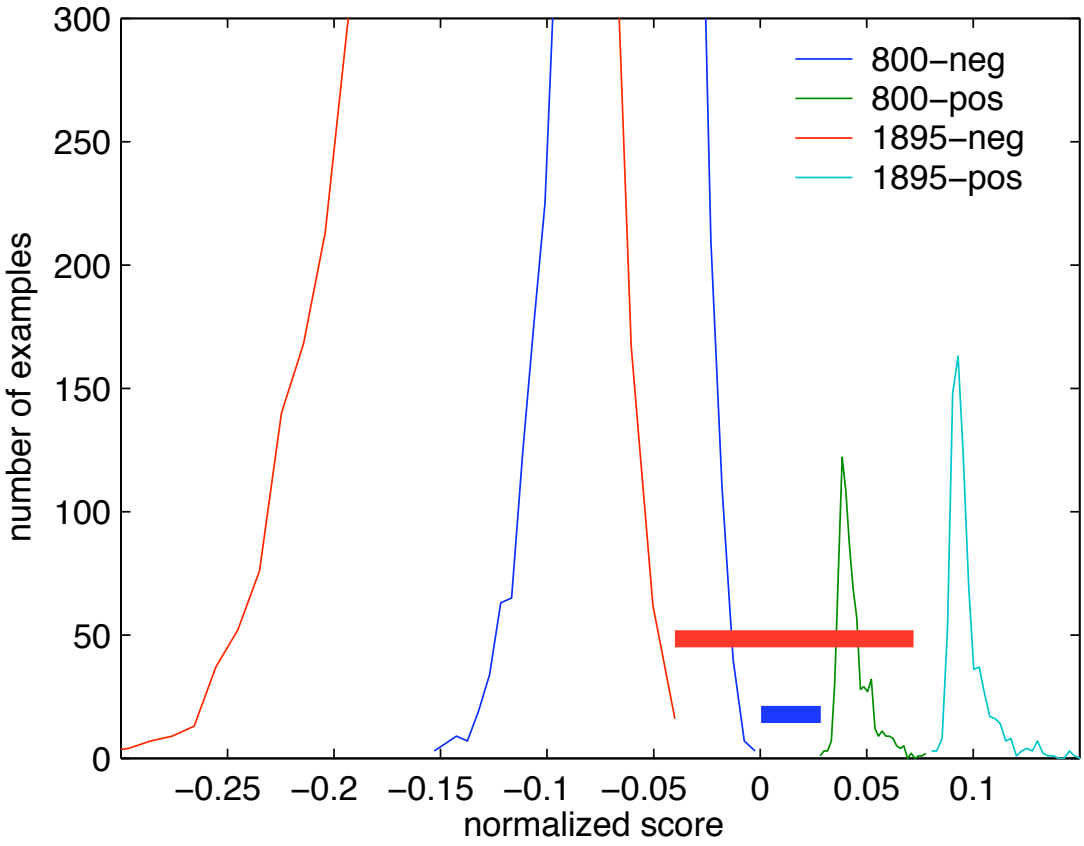


FIG. 3.46 – The distribution of training set scores at step 8 as a function of the boosting cycle.

see these outliers in figure 3.44 in the line of step 8.

In figure 3.52 we see the same kind of histograms for the learning set. We see that, of course, the learning set is totally separable and that after 300 weak learners there is already a clear line that can totally separate positive and negative examples. This, of course, makes sense, and has also a theoretical justification. It was proven (see [Freu 95]) that AdaBoost always reaches the point of separation for the learning set.

3.7.7 Conclusions and future work

We have presented a method for graduate collection of high quality training examples. In the results of our experiments, it is clearly seen that the examples we collect are becoming more and more hard to separate. In the experiments, we did not use all the input data which were available. It is interesting to observe the development of the process in further steps.

False positives generated in mature steps can guide us in choosing which features to add in order to further improve the detector's accuracy. These might be more complex features such as wavelet features as used by Viola and Jones [Viol 03], color based features, or movement based features. While these features are more computationally expensive, we can use the

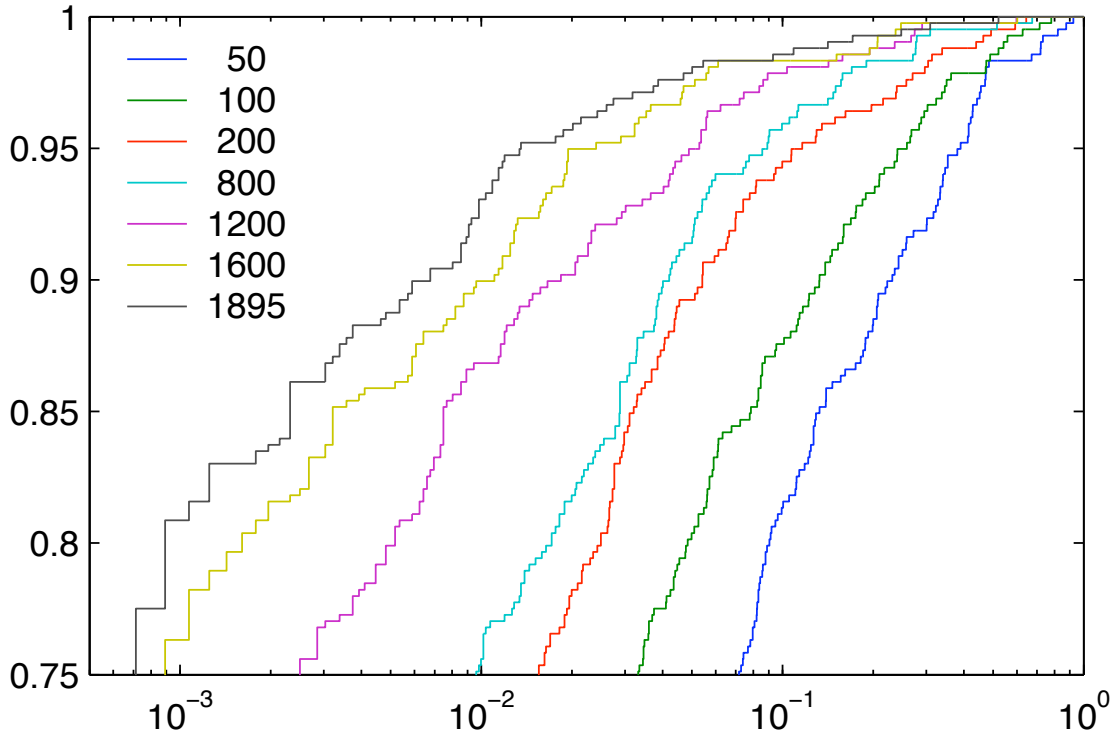


FIG. 3.47 – The validation-set ROC of the detector generated in step 8 as a function of the boosting cycle.

cascading method of Viola and Jones to restrict their calculation to very few cases and thus have small impact on the average time calculation while still achieving high accuracy.

3.8 Comparison between various camera types

This section brings quantitative results for pedestrian detection in various camera types. Results are brought for three types of camera : normal camera at daylight, near-infra-red (NIR) and far-infra-red (FIR) cameras at night. For each of these types we provide the curve of detection rate versus false detection rate (ROC curve), and the speed of operation.

3.8.1 Training details

In these experiments we used training and validation data. It will be noted that the sequences used for validation are never the same ones from which the learning examples are taken ; nor they contain the same pedestrians.

The training for all camera types was done on single-CPU Pentium-4 2.4Ghz desktop computers. For each type of camera, training time was in the order of one week for the cascaded and 3-4 days for the non-cascaded detector.

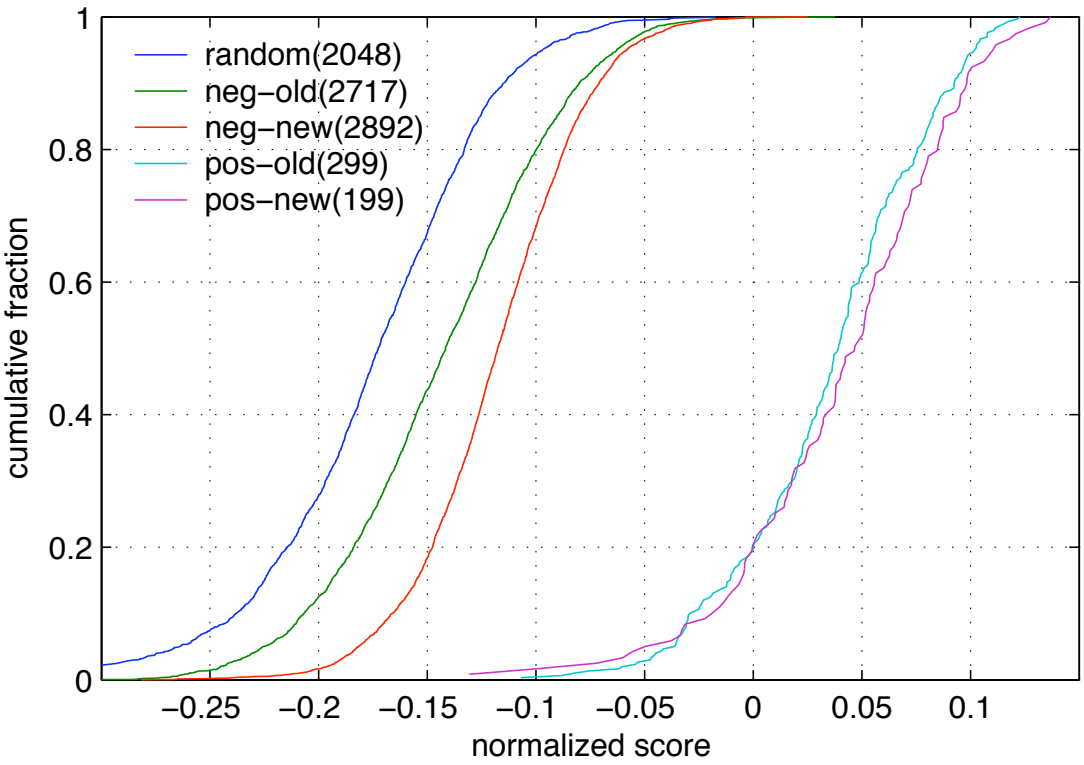


FIG. 3.48 – The cumulative distribution of the scores for examples in the validation set collected on step 8 versus examples collected in earlier steps.

3.8.2 Regular camera at daylight time

We trained a detector for pedestrian at daylight time, using a regular camera. For this training we used a mixture of video sequences : first, some video sequences taken in Versailles using a high-quality camera. Second, sequences taken in SmallVille[4] in the Parisian area, using a lower-quality camera. Examples for these data can be seen in Figure 3.53. The Versailles sequences had a resolution of 384x288 pixels, and the Smallville ones had a 640x480 resolution. To have an unified size, the Smallville sequences were downscaled to have a 384x288 resolution.

The training used 2428 positive and 22495 negative examples collected from these sequences, and validation ground truth containing 378 manual marking of pedestrians.

We carried out non-cascaded learning with 3000 features. Figure 3.54 shows the ROC curve for the detector.

3.8.3 Near Infra Red (NIR) camera

We trained a detector for pedestrian at night time, using a NIR camera. We used video sequences taken in Smallville, with a resolution of 640x480. An example of these sequences

[4]we are not using the real name of the town for reasons of confidentiality

can be seen in Figure 3.55.

The training used 2300 positive and 5600 negative examples previously collected from similar video sequences. For validation, we marked ground truth location of 145 pedestrians.

We carried out 3000-feature non-cascaded learning. Figure 3.56 shows the ROC curve for the detector.

3.8.4 Far Infra Red (FIR) camera

We trained a detector for pedestrian at night time, using a FIR camera. We used video sequences taken in Smallville, with a resolution of 640x480. An example of these sequences can be seen in Figure 3.57.

The training used 1647 positive and 3875 negative examples collected from similar video sequences. For validation, we marked ground truth location of 181 pedestrians.

We carried out 3000-feature non-cascaded learning. Figure 3.58 shows the ROC curve for the detector.

3.8.5 Conclusions

It's clearly seen from the graphs, that infra-red cameras see pedestrians at night better than normal cameras at daytime. This conclusion is not obvious at all. From the two types of infra-red, the FIR camera is clearly the best one.

3.9 Analysis of the cascade

The cascaded detector is an important contribution made by Viola and Jones. In their publications (e.g. [Viol 01]) they roughly state that the detection capability of a cascaded detector is about the same as a non-cascaded one. In this section, we would like to test that observation and explore this issue.

We will first compare the 2D ROC curve (that is, the conventional ROC curve) of a cascade with that of a non-cascaded detector. Later on, we will insert also the time dimension, and compare performance as well.

For the following experiments, we used video sequences taken from a frontal camera mounted on a moving vehicle, and collected 1224 positive and 2067 negative examples of cars. The cars are always viewed from the back, as seen in figure 3.59.

We trained a 20-layer detector. The number of features in the layers are : 3, 0, 0, 1, 0, 0, 2, 0, 3, 0, 6, 4, 7, 10, 16, 66, 9, 35, 65 and 162. The layers with 0 feature are degenerate layers which do nothing; the code is optimized to ignore them[5].

In parallel, we trained a flat (i.e. non-cascaded) detector with 500 features on the same training set.

[5]The zero-feature layers are created because in the learning process, as explained in section 3.2, each layer gets a task to arrive to a certain detection rate and false detection. Some layers' task is already accomplished by the layers when the process arrives to them.

3.9.1 2D-ROC analysis

In figure 3.60 we see a ROC multi-curve of each layer of the cascade, together with a ROC-curve of the non-cascaded version. Let us concentrate on the cascaded part first. This complex graph is prepared as follows :

- **Step 0** Layer 0 (the first layer) is used to create a standard ROC curve. Its threshold is being driven from 0 to 1 while the combinations of detection-rate and false positive are being plotted. Note that detection rate is in the range 0 to 1 (the number of detected objects divided by the number of objects), and the false positive rate is in the form 1 :N, meaning one false detection for each N sub-windows examined.
- **Step 1** A cascaded detector is being composed of layer 0, whose threshold is being fixed on the one that came out of the learning process, and layer 1, whose threshold is being driven from 0 to 1 in a similar manner as in the previous step. Again, combinations of detection-rate and false positive rate (using the combined 2-features detector) are being plotted.
- **Steps 2-19** We continue with this process for the rest of the layers. That is, in step k a cascaded detector is being composed of layers 0 to $k-1$, the threshold of these layers being fixed on the ones that came out of the learning process, together with layer k, whose threshold is being driven from 0 to 1. Again, combinations of detection-rate and false positive rate (using the combined k-features detector) are being plotted.

The envelope of this multi-curve reflects the actual performance of the cascade. These are the best detection rates that can be obtained for any given false detection rate (or vice versa). Now, we can compare this performance with that of a non-cascaded detector.

Let us consider now the ROC curve of the non-cascaded 500 features detector, whose ROC-curve is also presented in figure 3.60. We can see, that **the detection of the non-cascaded detector is much better than that of the cascaded one**. This conclusion is not surprising : if a sub-window was rejected by one of the layers in the cascaded detector, it will not appear as detected. This will reduce the detection rate, because this sub-window could have been approved by one of the later layers, if that layer had seen it. In the non-cascaded detector, such a sub-window is detected because it passes all the features.

In the next subsection we will see that, when taking into account the efficiency data (i.e. processing speed), the advantages of the cascade are fully seen.

3.9.2 3D-ROC analysis

The results shown in the previous subsection are not complete, if we don't mention the time domain.

In figure 3.61 we see a 3D graph that represents the ROC curve of the non-cascaded detector as a function of the number of classifiers used. This number has a direct (and linear) impact on the detection speed.

The average number of classifiers applied on each sub-window in our **cascaded** detector is around 5. Therefore, to perform a real comparison between the cascaded version and the non-cascaded one, one should place the ROC multi-curve of the cascade near slice number 5 of the 3D ROC curve shown in figure 3.61. However, by observing the 3D ROC curve,

we can see that the cascaded detector has detection capabilities that are similar to that of the non-cascaded detector with 300-400 features, and surely much better than a 5-features non-cascaded detector.

3.9.3 Conclusions

It is quite clear that a single-layer 5-features detector cannot efficiently detect vehicles. However, its performance is similar to that of the cascade, because in the cascade, **most of the sub-window filtering work is done by the rest of the layers without significant time consuming**. This is the big innovation of the cascade.

However, when compared with a full non-cascaded detector (with 500 features), **the detection of the cascade cannot reach the same level**. This is a certain price to pay for the high speed.

3.10 Conclusions

In this chapter we have reviewed and contributed to the domain of visual object detection using machine learning. We have presented the state of the art and shown how previous works used AdaBoost for object detection. Then, we have introduced our own types of visual features, which were designed to better fit the needs of on-board automotive applications.

We have presented the 5x5 visual feature, which was specially optimized to work with the CAMELLIA project prototype. We have presented also the control-points features, that were designed to be highly efficient both in memory and time. We have compared the features with each other and with the work of Viola and Jones and shown that our features yield better results.

In particular, we have chosen to present the data in 3D-ROC curves, in a way that we feel reflects the best the real advantages of a system. The control-points features produce the best 3D-ROC curves on real-world data.

We have shown that the attentional cascade, suggested by Viola and Jones, is reducing the quality of detection. Again, viewing the data in 3D-ROC curves, shows that the cascade gives high speed benefit as opposed to the reduction in detection rates it causes.

We have chosen to concentrate on several additional points : one is the SEVILLE method we developed to collect new examples. This method makes it much faster to collect a good training set. The second issue if the comparison of various camera types. Among other conclusions, we have shown that Far Infra Red (FIR) camera at night time performs much better than a regular camera.

3.11 Future work

The research presented in this chapter can be further developed to many directions, which get out of the scope of this thesis. We hereby briefly review what can be done in order to improve the object detectors.

3.11.1 The image pyramid

To detect pedestrian of all sizes, an input image I is first preprocessed : the system creates a set of images $I^0 \dots I^k$ (so called a "pyramid" of images) where $I^0 = I$, and each I^i is created by warping I^{i-1} to 75% of its size. The detector than runs a constant 48x24 detection window on all these images, eventually detecting different scales of pedestrians.

Another way to implement the multi-scale search is to upscale the features, rather than downscale the image. In such a way we get rid of the need to create the image pyramid (see [Viol 01]). An adaptation of this method has to be done to our visual features. Such adaptation, however, is not trivial since one can scale regions, but not points... some intelligent process might be possible, however, to avoid the preparation of a full image pyramid.

3.11.2 Spatial density

From various observations, it seems that detections of pedestrians usually come in groups, as opposed to false detections, which come separately. See Figure 3.62. One might be able to take advantage of this phenomenon to throw away false detection. This work is also not trivial ; a full analysis must be carried out to see how this information can be used and if it, indeed, brings better detection rates.

3.11.3 Tracking (temporal density)

The same phenomenon exists also in the temporal axis : a pedestrian which was at position X in one image, is likely to be around that position also in the next image. Therefore, the system has to validate only pedestrians which were detected several times around closely-related positions.

In further chapters of this thesis, we implement a variant of this method. We use particle filtering to track detections created partly by AdaBoost.

3.11.4 Low-level motion

It was found [Viol 03] that it can help to take into account motion data between two subsequent images. To accomplish this, we allow the weak classifiers to examine not only the image I_k, but also the difference image $I_k - I_{k-1}$. It turns out that pedestrians have a very special movement, especially in the area of the hands and legs.

3.11.5 Limited search area

We can tune the system to detect only pedestrians with high confidence, then run it on many hours of video. Then, the system will have all the possible positions where pedestrians can be found. Having this "mask" in hand, we can later limit the search only to these positions. This will not only accelerate the system, but also will omit false detections which could have been created in the "excluded" positions.

This operation can be considered as learning the camera parameters automatically.

3.11.6 Context learning

A recent work [Fink 04] shows that it is worthwhile to apply AdaBoost also on the relations between objects. In such learning, the system learns to identify pedestrians together with other outdoor objects such as cars, trees, sidewalks and crosswalks. Then, the system uses some weak rules which have information on the relations between the objects. This way, for example, a pedestrian could never be detected above a car, or on a tree (although the later is possible in special situations...).

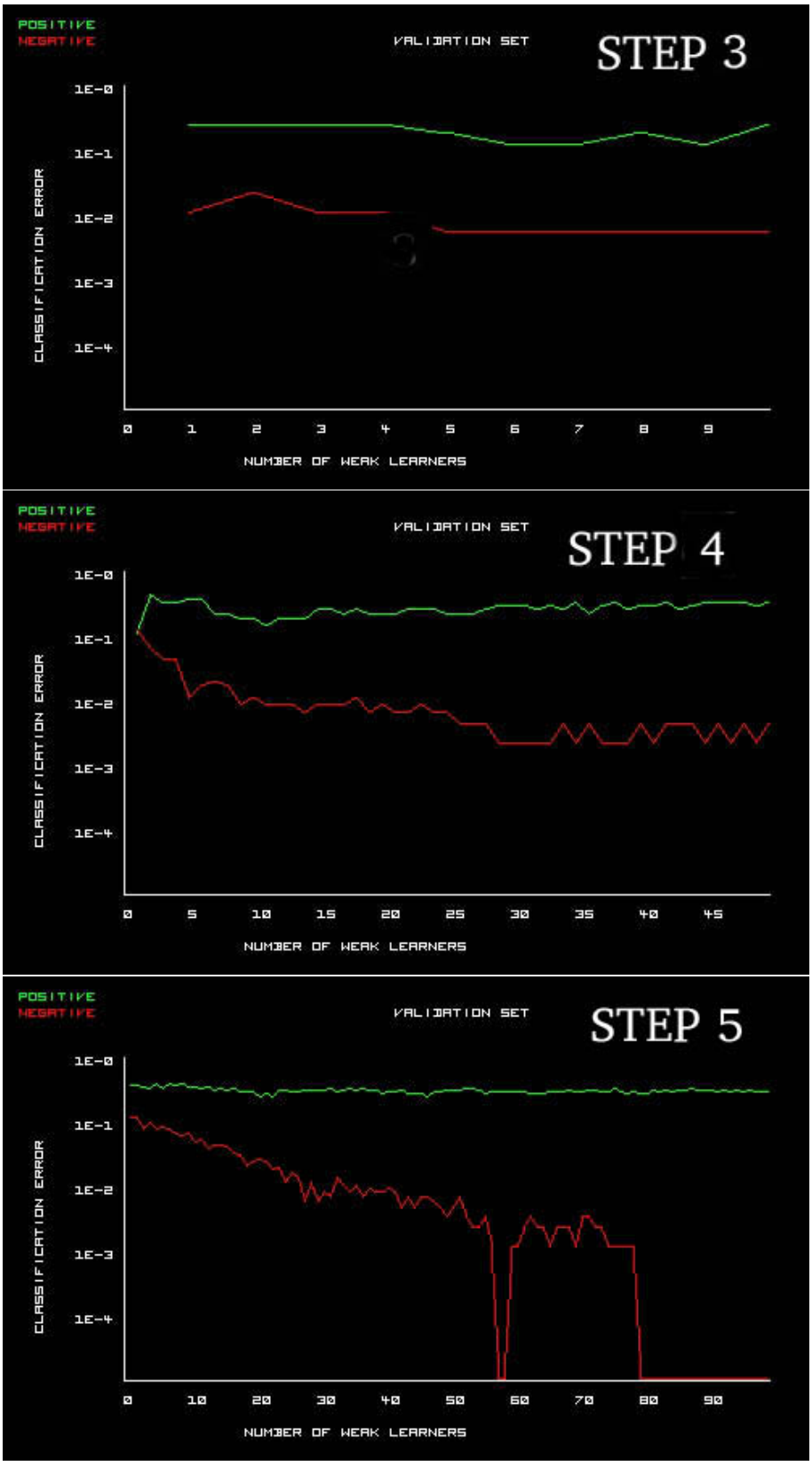


FIG. 3.49 – Learning graphs for steps 3-5.

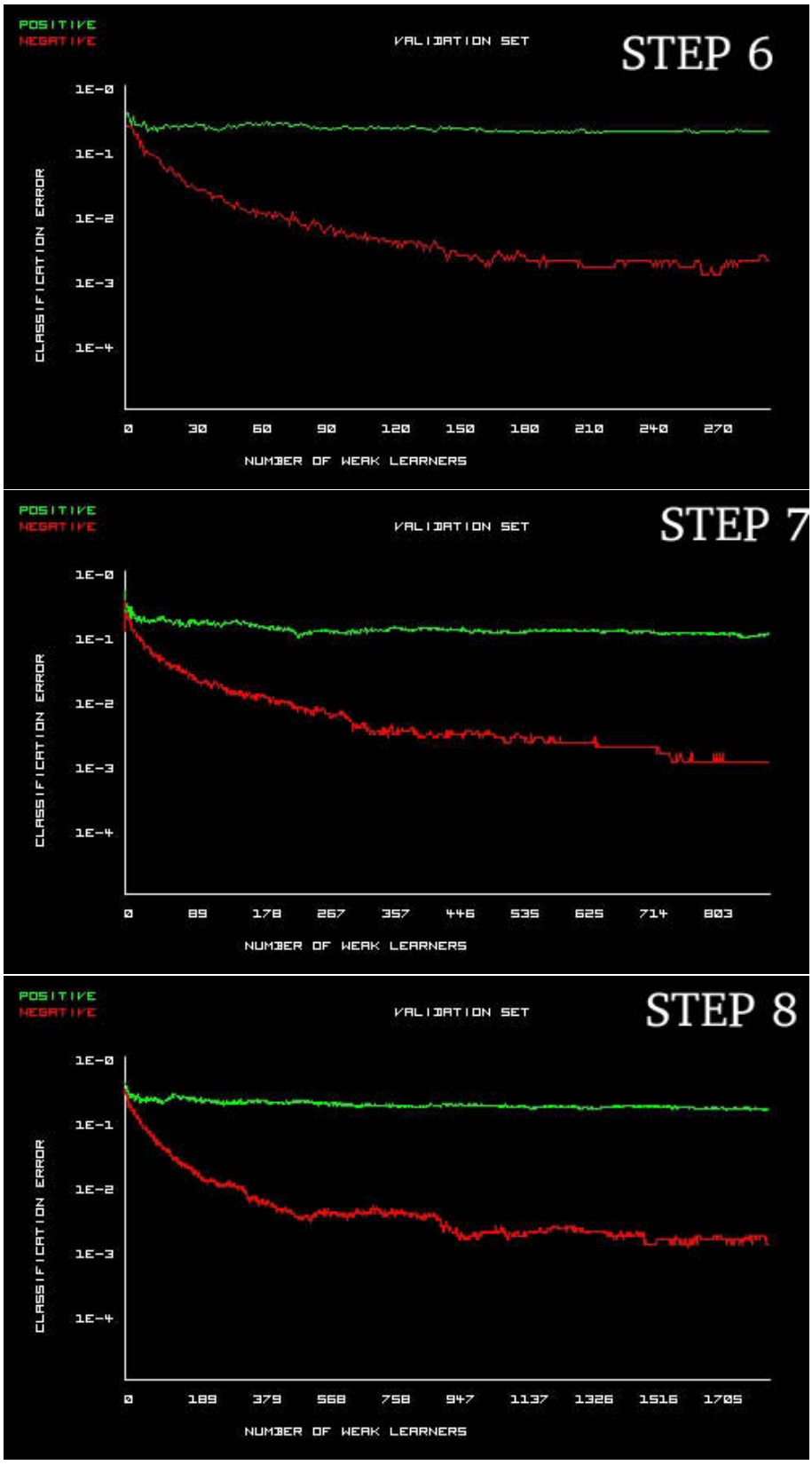


FIG. 3.50 – Learning graphs for steps 6-8.

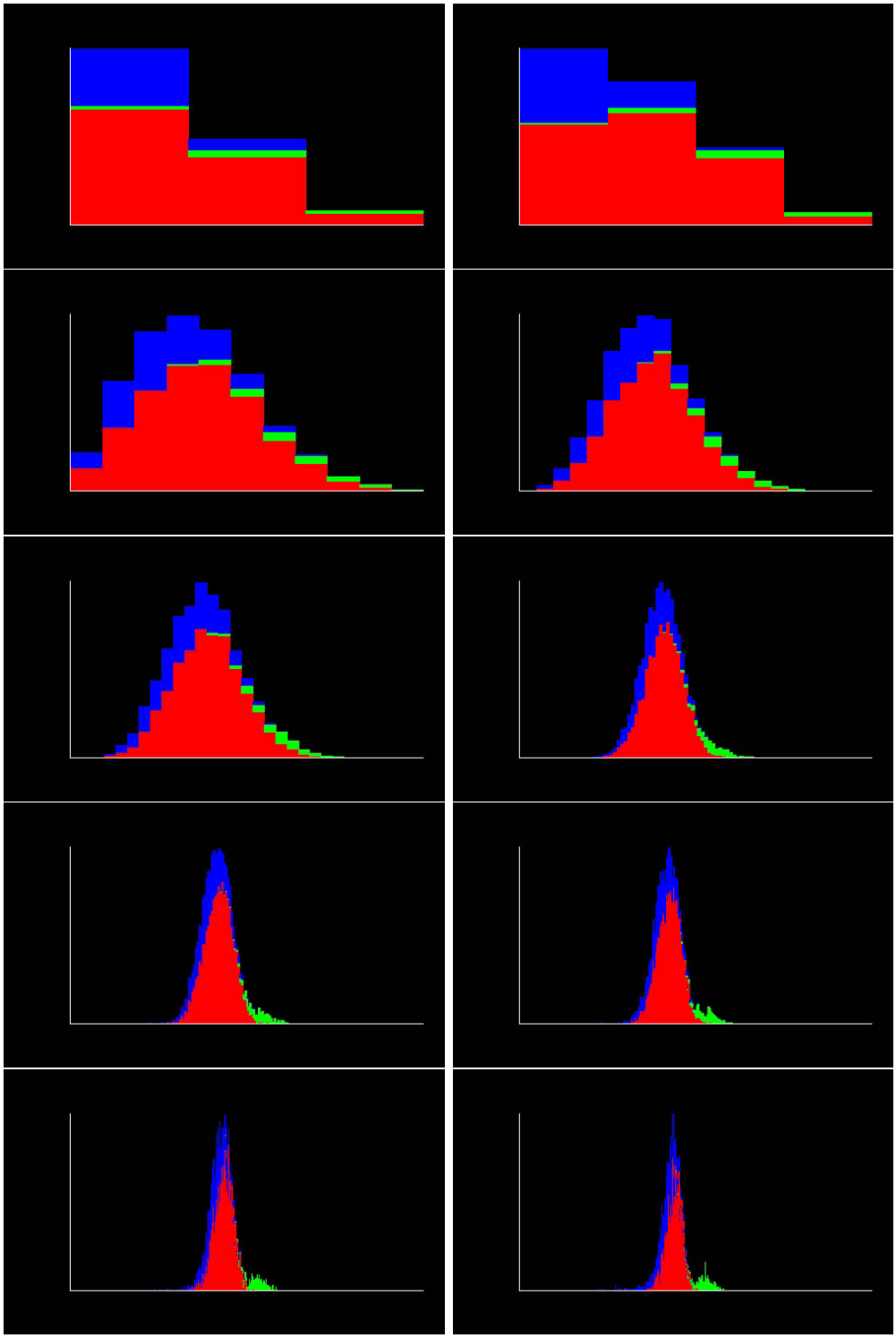


FIG. 3.51 – Boosting values histogram for the validation set in step 8, for different numbers of weak learners. From left to right, top to bottom : 2,3,10,20,30,100,200,300,1000,1895 weak learners.

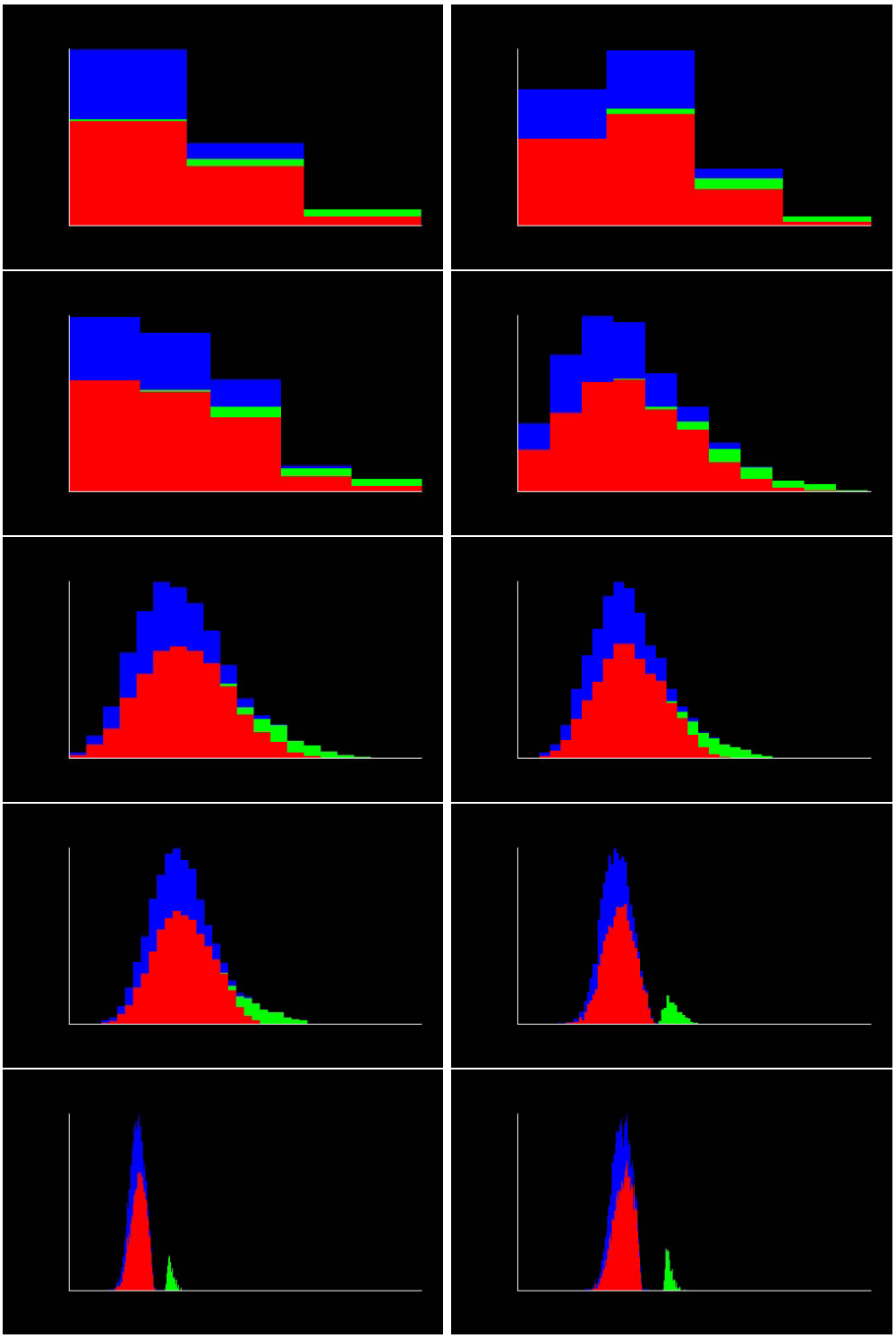


FIG. 3.52 – Boosting values histogram for the learning set in step 8, for different numbers of weak learners. From left to right, top to bottom : 2,3,4,10,20,30,40,100,200,300 weak learners.

FIG. 3.53 – Images from the video sequences used for the training of pedestrian detection at daytime : the Versailles sequences (top) and the Smallville sequences (bottom).

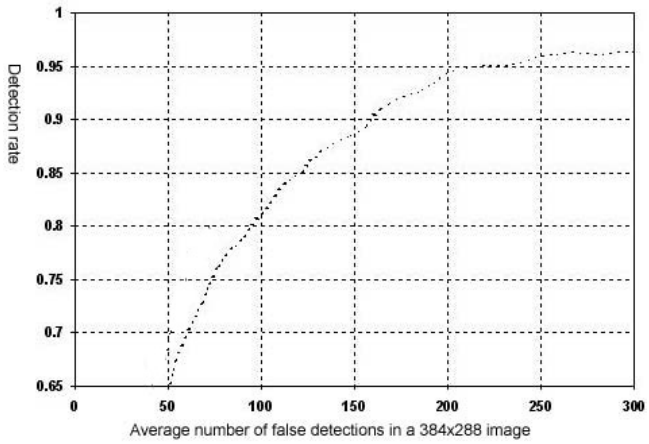


FIG. 3.54 – ROC curve for daylight pedestrian detection with a regular camera.

FIG. 3.55 – An image from the video sequences used for the training with the NIR camera.

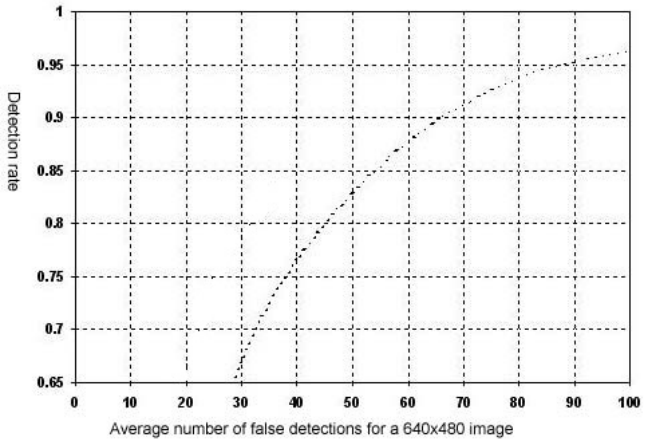


FIG. 3.56 – ROC curve for night-time pedestrian detection with the NIR camera.

FIG. 3.57 – An image from the video sequences used for the training with the FIR camera.

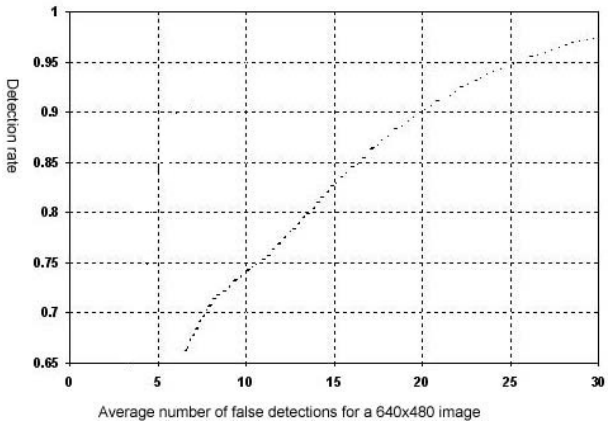


FIG. 3.58 – ROC curve for night-time pedestrian detection with the FIR camera.

FIG. 3.59 – Examples of cars used for the learning.

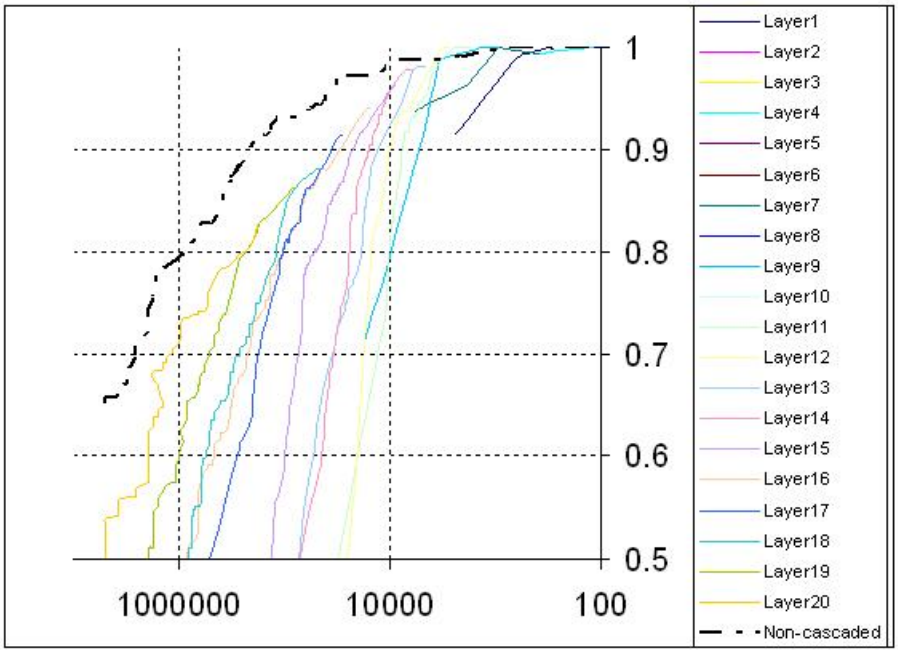


FIG. 3.60 – The ROC multi-curve for the 20-layers cascaded cars detector. The x axis measures the false detection rate in the form 1 :N (one false detection for each N examples examined). The y axis measures the detection rate (number of detected objects, divided by the number of objects). In dashed line, is shown the ROC-curve of the non-cascaded 500-features detector. The non-cascaded detector is clearly superior.

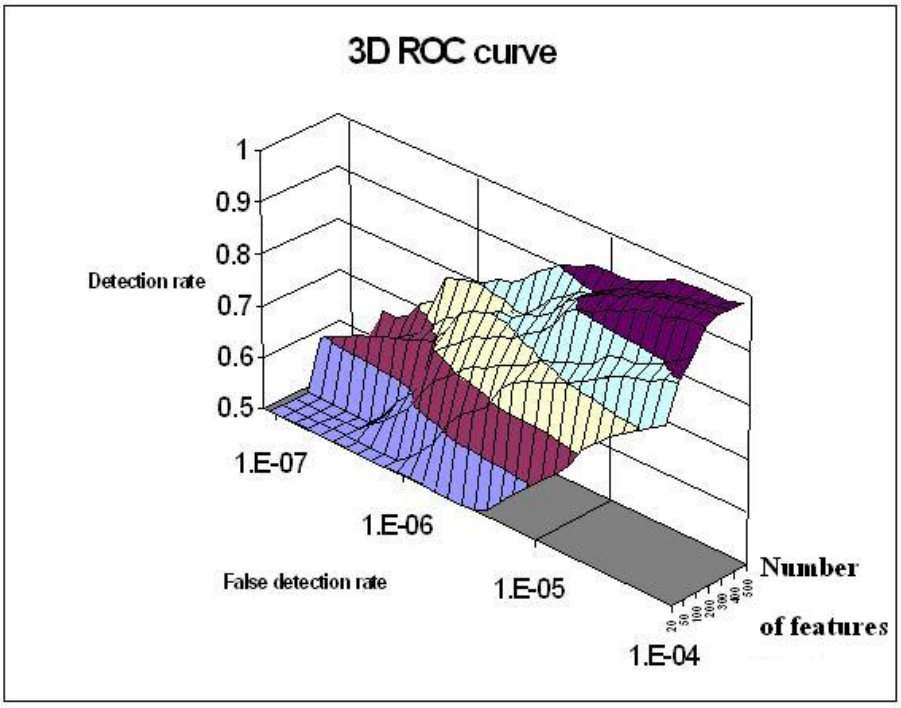


FIG. 3.61 – The 3D ROC graph of the non-cascaded detector.

FIG. 3.62 – The dense nature of detections. Note that around the pedestrians, several detections are formed, whereas the false detections stand alone.

Chapitre 4

Motion estimation algorithms

4.1 Background

This chapter describes an algorithm for detection of object motion in video sequences. The algorithm was originally based on an algorithm described by Wittebrood and de Haan [Witt 01], yet it has undergone several important changes. The newly implemented algorithm forms the basis of Camellia high-level applications, and will be called hereinafter CMSA (Camellia Motion Segmentation Algorithm). It is also used in the Stop&Go system described in chapter 6.

The basic method employed in the CMSA algorithm is motion estimation (ME). This method is taken from the field of video compression. Roughly speaking, ME tries to detect objects in the video scene, which are moving. The input of the algorithm is a video sequence and the output appears in two levels : a map of movement directions in the lower level, and a set of detected moving objects in the higher one.

4.2 General overview of the algorithm

4.2.1 Motion models

A *motion model* is a set of numbers expressing the nature of a movement of an object in the scene. Motion models can appear in various levels to model different kinds of movements : the basic level introduces two numbers which express the 2D movement of an object on the screen. A higher level models also zooming, for which it uses additional two numbers for the zoom center and another number for the zoom extent. The article [Witt 01] uses two numbers for the zooming extent, which allows also bodies to zoom differently in the X and Y axis (this gives 6 numbers altogether). Further enhancements can model rotations, deformations, etc.

At this version of the algorithm we chose to limit ourselves to 2 numbers only. A look at the result shows, that using only 2D movement, plus some adjustments, gives reasonable results. We believe that using more than 2D models will complicate the situation to a large extent because it will add more dimensions and thus dramatically enlarge the space of the possible models. This overhead will take more than it will give.

The motion model is a fundamental notion in our algorithm. What we try to do in the algorithm, basically, is to match a motion model to each object in the image (and in the video sequence).

4.2.2 Other definitions

A *block* in a video image is a 16x16 pixels block. This is the basic unit for analyzing the image.

Given a block in an image and a motion model, we can check if this motion model is a good motion model for that block. What we do is simply compare the block in the current image, with the same block, translated and zoomed, in the previous image. We compare pixel by pixel and calculate the Sum of Absolute Differences (SAD). The lowest the SAD is, the better the motion model is. And indeed, in the rest of this paper we will use the term of good motion model (with respect to a specific block in an image) to express a motion model that yields a low SAD. It will be noted that SAD is typically a task to be performed by hardware. In particular the CAMELLIA low-level algorithms include SAD computation.

4.2.3 Basic diagram

Figure 4.1 gives a diagram that summarizes the operations of the algorithm. As seen in the diagram, the algorithm runs on an internal and external loop.

The internal loop (bounded in dotted line) runs on all the blocks of a single image, taking block after block and checking different motion models to find the best one. It involves creating sets of candidate motion models for checking. The external loop is running on the different images of the video sequence ; for each image it creates a map of motion models, and then tries to group together blocks of similar motion models. The last step is forming these groups into objects, maintaining confidence for each of them. The following sections describe in details these steps.

4.3 Main (internal) loop on blocks

The inner loop of the algorithm runs on blocks. The image is divided into blocks. We repeatedly run on the image blocks, passing the image several times. Our goal in this loop is to find, for every block, a good motion model. That is, we would like ideally to find the exact same block in the previous image in the sequence, may be moved. If we could do this, then this translation is a perfect motion model for this block. In real life, we rarely find the exact same block in the previous image, but many times we find the same block, moved - and slightly changed. If this change is indeed low, then this moving forms a good motion model for that block.

4.3.1 Fetching of new blocks

At the beginning we have tried to use sophisticated mechanism which tried to bring each time a block that is most suitable for handling. In practice we found that the best way is

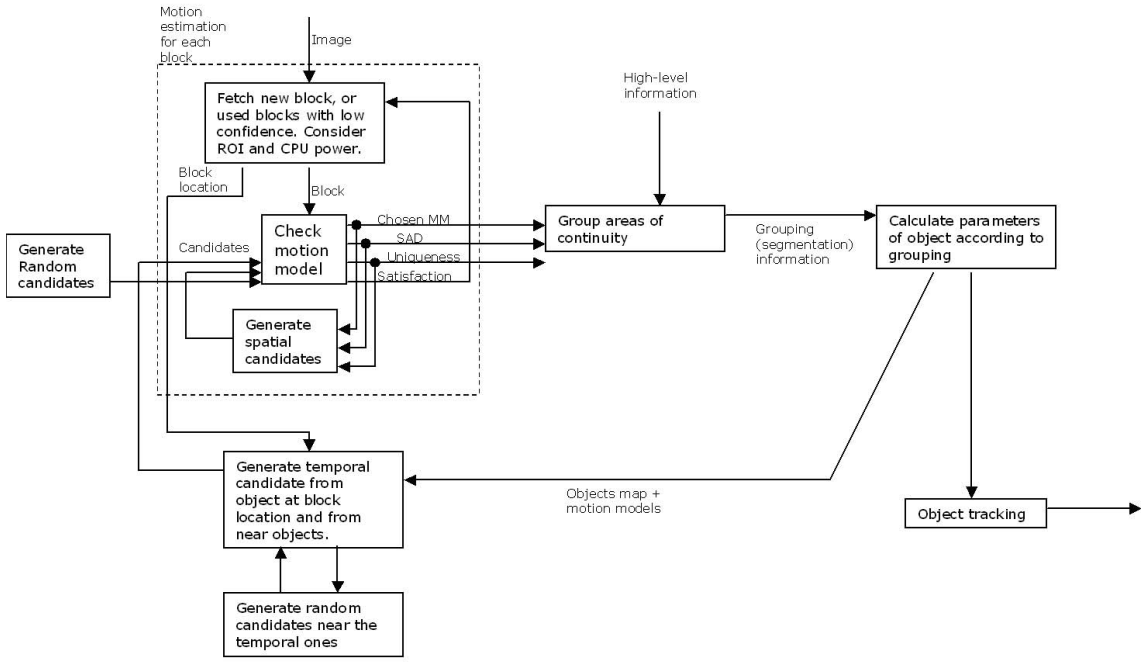


FIG. 4.1 – The algorithm component diagram.

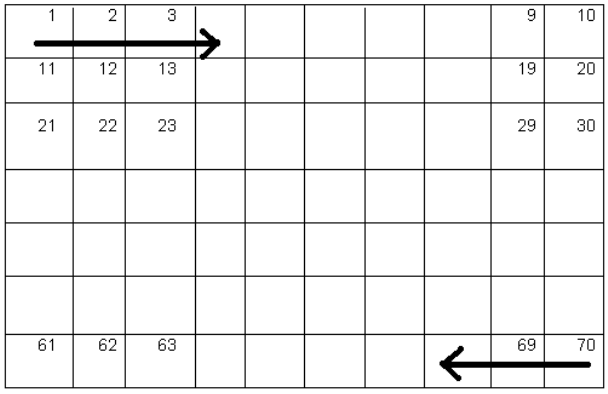


FIG. 4.2 – Different directions of scanning

to scan the image in alternating directions and just pass on all the blocks two times, as demonstrated in Figure 4.2. The 2 directions are (according to the numbers in the figure) :

- $1, 2, 3, \ldots 9, 10, 11, 12, \ldots 69, 70$
- $70, 69, 68, \ldots 62, 61, 60, 59, 58, \ldots 3, 2, 1$

This is enough for the diffusion of the spatial candidates within the image : the first will diffuse spatial candidates down and right, the second will diffuse them up and left. Also note that in the second pass, the system will not take blocks, which are considered as "done". Among these we can count non-unique blocks (see further), or blocks for which we tested many motion models already (we will use some upper bound on the number of possible checks).

These two passes can be repeated if a stronger diffusion is needed. For example, in the image stabilization application of CAMELLIA (see [Abra 04]), this process is repeated 16 times. After these passes a special process, called uniqueness contradictor, is applied (see subsection 4.3.7).

4.3.2 Motion-model checking

The motion model checking is the heart of the algorithm. The idea here is to use a set of candidate motion models and to check if any of them is a good motion model for the current block. The algorithm is getting as input the following data :

- a block in the image (the algorithm will access the block's pixels in this image, as well as pixels in the previous image) ;
- several candidate motion models - some taken from already-analyzed neighboring blocks (spatial candidates), some taken from already analyzed blocks in the previous image (temporal candidates), some randomly generated (random candidates), and some deliberately generated (initiated candidates). See further sections about the generation of these candidates.

The algorithm is taking each candidate motion model and checks if this model is a good model for this block. This is done by calculating the SAD for all the pixels in the blocks, compared to the previous image. The algorithm is giving the following outputs :

- a motion model, chosen from the candidates, which yields the lowest SAD ;
- the SAD value for that selected motion model ;
- the uniqueness of the SAD among the SAD of other candidate motion models.

The uniqueness value is explained in subsection 4.3.4.

4.3.3 The actual SAD calculation

The input of the algorithm comes as a YUV image. The SAD is first calculated on the Y component of the image. Then, the U and V components are sub sampled in a ratio of 1 :2 (for each axis), and the SAD is computed on the resulting 8x8 blocks of the U and V image components. The resulting SADs of the Y, U and V are added with weights of 50% for the Y, 25% for the U and 25% for the V. These weighting is intuitively conforming with the size of the blocks ; further experiments have to be conducted in order to find the optimal weighting. This scheme conforms to the 4 :2 :2 format.

It is important to note that sometimes it is useful to compare each image not with the image before it, but with N images before. This is what we do in the vehicle application : we compare with 3 images before. That is, if $G_1, G_2 \dots G_N$ are a sequence of video images, then G_4 is compared with G_1, G_5 is compared with G_2, and so on. This gives a better signal to noise ratio since it sums the real movement, while continuing to have the same amount of noise. In image stabilization we simply compare with the previous image, since movement there is, by definition, not smooth.

Why not more than three ? It was seen that using more than 3 images back gives unwanted effects, as :

- Movement of objects is no longer linear.
- Motion model is appearing after too long delay.

In addition we have to consider memory limitations. This is why an offset of 3 was chosen.

4.3.4 The uniqueness of a motion model

Experiments have shown that, in addition to the quality of the SAD of a motion model (in relation to a specific block in a specific image), there is another important value, which is the uniqueness. To understand the uniqueness value, consider the example of a smooth object, like the sky. Inside this object, you will find many blocks, which will match many motion models with a good SAD. If you take a block in the sky and move it left, it will match. Move it right, it will also perfectly match. Move it up, it will match again. This effect will appear many times inside objects, even ones that are less smooth than the sky. Figure 4.3 shows how blocks on the edges of a car are giving duplicate motion models.

Note, however, that in some blocks we might have several different motion models, which all match with good SAD, but all directed in the same approximate direction. Usually these models have a difference of 1 pixel. It will be a waste to decide that these are non-unique motion models and not use them for the analysis, because this set of models do express the true motion of the object. Hence we should be careful before we "declare" that a motion model is not unique and subsequently abandon it.

We need, therefore, to define what uniqueness is. We give the following set of definitions :

Two models are considered *close* if their corresponding movement vectors are close :

- with respect to direction (angles are close within a minimum angle α^{min}) ;
- with respect to vector length (lengths are close within a minimum distance L^{min}).

Actual optimal values for the last two constants were found in experiments and are $\alpha^{MIN} = 0.6$ (in radians), $L^{min} = 1.0$. In addition, it might be better not to take into account the angle distance, if the lengths of both vectors are very small. This however was not addressed up to now because we are not grouping the zero motion models.

Consider a block B in a specific image, in a video sequence. Say that for this block, a set of motion models $m_1 \dots m_N$ has been already found, and assume that this set is ordered from good (low) SAD to bad (high) SAD - that is, m_1 is the best motion model.

We say that m_1 is the *selected motion model* of the block B. If $N = 1$, then m_1 is absolutely unique and has a uniqueness value of 100. Otherwise, it has a uniqueness of :

FIG. 4.3 – One object contains several 2D motion models.

$$Uniqueness(m_1) = \min_{m \in S} \frac{SAD(m)}{SAD(m_1)} \tag{4.1}$$

where $S \subseteq \{m_2, \ldots m_N\}$ are all the rest of the motion models which were found, and which are :

- pair-wise not close to each other,
- pair-wise not close to m_1.

and $SAD(m)$ is the SAD value of a motion model m with respect to the block B.

An uniqueness of 1.0 means, that there is another motion model, substantially different, which gives the same SAD value. This is clearly the lowest uniqueness. A SAD of 2.0 is considered to be a quite good uniqueness. Actual values for defining which block has a unique motion model are moving between 1.5 and 2.5, depending on the application. In the actual algorithm we hold for every block an array with several motion models, up to a fixed number of models (currently 3). We hold them sorted by the SAD, from low to high. This set is actually giving the whole picture about the status of this block. When a new result (a motion model + SAD value) arrives, we run on the existing models and we might insert the new result, depending on its SAD. Note that if the array contains a model which is close to the new result, but with worse SAD, it will be replaced. If it's with a better SAD, the new result will not get in.

4.3.5 Minimal SAD values

Experiments have shown, that it is essential to fix a minimum value of SAD, under which formula 4.1 in section 4.3.4 is not active. For example : if a specific motion model yields a SAD of 2.4 pixels, and another model yields a SAD of 0.8 pixels, we could conclude, according to formula 4.1, that we have a uniqueness of $\frac{2.4}{0.8} = 3.0$. However, both values, 0.8 and 2.4, are very low and can be considered "good" SAD values. Therefore we must fix formula 4.1 so it will determine, in such case as our example, that the uniqueness is low. All we have to do to achieve this is define an extended SAD function :

$$SADextended(m) = \max(SAD(m), \hat{\mathcal{S}})$$

where $\hat{\mathcal{S}}$ is a constant that expresses an extremely "good" SAD. Actual experiments have shown that a good value for this constant is ranging between 1.0 and 2.5, depending on the application.

4.3.6 Spatial candidates generator

Generating the spatial candidates is a simple task that, for each block, takes the motion model of its neighboring blocks which already have a calculated motion model, and uses them as a candidate motion model for this block. The current provided version uses 8 spatial candidates in each of the 4 scans. This is likely to be improved without loss of performance to 2 spatial candidates in each of the 2 scans.

4.3.7 Initiated candidates generator (uniqueness contradictor)

In actual experiments, it was shown that the program tends to miss many cases of non-uniqueness and attach to some blocks a good motion model which it believes to be unique. A typical case would be on the edge of an object - the chosen model seems to be unique, but the opposite of this model is also giving good SAD. However the opposite model was never checked, because it accidentally did not diffuse through the random-spatial-temporal candidates system.

This phenomenon causes many artificial effects which appear as noise in the output. A good method to solve it is to run a complementary process, after the process of motion model finding, which tries to contradict the fact that a block contains a unique motion model.

Given such a block with a motion model $\langle m_X, m_Y \rangle$ which is believed to be unique, we perform a SAD test with the following motion models :

- $\langle 0, 0 \rangle$
- $\langle -m_X, -m_Y \rangle$

Testing with a zero motion model usually helps when a block is inside a smooth object, was accidentally tested on a small number of non-zero motion models, and one of them happened to be good. By testing with the zero model we are likely to get a SAD, which will render the current model non-unique. This is because usually inside a smooth object most of the motion models have similar SAD.

FIG. 4.4 – The car entering on the right appears only by its exterior. The interior blocks are not unique, therefore are not considered as having valuable information.

By testing with an opposite motion model we are contradicting cases of long edges (e.g. the side of a car) which tend to falsely conclude that they are moving in one direction of the edge. By testing the other way of the edge, we can see that this block is not unique.

4.3.8 Weights for different types of candidates

In [Witt 01] it is noted that different kinds of candidates should be weighted differently. Indeed, we used this conclusion and we weighted our candidates. Actual experiments show that the SAD of a motion model should be multiplied by :

- 0.85 - if it is a temporal candidate
- 1.0 - if it is a spatial candidate
- 1.1 - random candidate
- 1.0 - initiated candidate

4.4 External loop on images

4.4.1 Grouping of areas

After the inner loop has finished working on an image, we have a motion model for every block. Our goal now is to group blocks that belong to the same object.

In principle, the blocks of a single object are moving together ; therefore we expect, inside an object, to find high continuity of motion models with low SAD. To find the bodies of this continuity (the objects), we employ a BFS (Breadth First Search) algorithm on the blocks.

The algorithm starting blocks are blocks with the highest uniqueness, to increase the quality and performance of the grouping. The algorithm continues from a block B to its neighbor if :

- the neighbor is unique,
- the neighbor has a sufficient SAD,
- the neighbor has a similar motion model to B.

Coming to analyze this process, we can notice the following cases :

1 The classic case is where a different object is detected. The neighboring block will have a totally different motion model, good (low SAD) and unique (high uniqueness). We do not continue with this block, because it probably belongs to a different object.

2 We might hit many neighbors with high SAD, with which we will also not continue.

3 Another interesting case is the interior of smooth objects (like the sky). A typical block in this area would have a good motion model which is not unique, because it can be moved according to many motion models and still yield low SAD. We do not include these blocks, so many objects indeed appear only by their exterior (see Figure 4.4).

4.4.2 High level information in grouping

This is the place in the algorithm to plug in external, high-level information that will help to group the objects. If, for instance, we know that every too lights ordered horizontally are the lights of a car, we can group them together even though the low-level algorithm will not do so. This grouping can later help to produce better candidates of motion models.

4.4.3 Minimal motion value

It is clear that we don't have to assign every block to an object. First of all, blocks without unique, good, motion model are not assigned. In addition, in a typical image, the background forms a big and non-continuous "object", which we should not try to group. To achieve this we defined a minimum movement threshold, under which we are not considering a block as belonging to a specific object. In the sequences with this report this threshold is 0.33 pixel per frame.

4.4.4 Smoothing of objects

On the borders of objects we are very likely to get a lot of single-block objects. In these regions there will be a lot of blocks which will have poor motion models - that is, motion models with high SAD. Most of them are not real objects but artifacts of the algorithm. They will be cleaned by smoothing the map of objects up to a certain threshold that defines what is the minimum size for a real object is. Actual experiments show that only objects of 2 blocks and above are considered.

4.4.5 Parameters calculation

Once we have a segmentation of the image to objects, we want to compute for each object its unified motion model. An object is moving uniformly ; it was detected during the grouping

phase because it had similar motion model between its blocks ; now we want to compute the motion model of the object.

The computation is trying to find a motion model such that it will minimize the sum of SADs on all the blocks of the object, according to that model. There are several options, which can be implemented. Here they are, from complex to simple :

- To use a gradient search to find a good model, in each step of the search to re-compute the sum of SADs to all blocks in the object ;
- To give the average motion model of all the blocks in the object ;
- To give the median motion model of all the blocks in the object ;
- To give the motion model of the starting block.

We chose the last way, which is giving the motion model of the starting block. The starting block was chosen by the grouping process to start a new object because it had the highest uniqueness, therefore it can be considered as a good representative of the block's motion model. This is also obviously the fastest way.

4.4.6 Temporal candidates generating

Once an image is ready, i.e. the objects in this image have been detected and every one of them has a motion model, we can use these motion models as candidates to the analysis of the next image. This is one of the most important parts of the algorithm - it is based on the fact that the next image is very likely to contain the same objects or almost the same objects, hence their motion models can be used.

To produce temporal candidates we take each block, and take its motion model from the previously analyzed image. Note that as opposed to the SAD comparison, we do not take the motion model from 3 images before, but from the very last one.

4.4.7 Object based candidates

Instead of simple temporal candidates we can expand to model-temporal candidates. This means that we do not take the motion model of the same block in the previous image, but we take the motion model of the object, to which the same block in the previous image belongs. This is the core element in the idea of "object based motion segmentation". Once we will use other techniques to refine the shape of the object - as edge detection - we will improve the quality of this model-temporal candidate motion model.

4.4.8 Randomization

The algorithm uses a strong element of diffusion. Therefore, if none of the spatial or temporal candidates are relevant, we need a small component of randomization in the generation of the candidates. We implement this by generating more candidates, in addition to the spatial and temporal candidates. We do it as follows :

- Defining a "motion span" distance $\bar{\mathcal{D}}$ - a range of motion we would like to be able to detect, measured in pixels per second[1].

[1] This value is 5 pixels in the FADE2 application, and 16 pixels in the CAMELLIA stabilization application,

- Generating a fixed number (normally 3) of candidates of the form $\langle m_X, m_Y \rangle$, where $m_X = rand(-\bar{\mathcal{D}}, \bar{\mathcal{D}})$ and $m_Y = rand(-\bar{\mathcal{D}}, \bar{\mathcal{D}})$.
- Using the sole temporal candidate $\langle m_X^T, m_Y^T \rangle$ to generate a fixed number (currently also 3) of candidates of the form $\langle m_X, m_Y \rangle$, where $m_X = m_X^T + rand(-\bar{\mathcal{D}}, \bar{\mathcal{D}})$ and $m_Y = m_Y^T + rand(-\bar{\mathcal{D}}, \bar{\mathcal{D}})$.

By $rand(a, b)$ we denote a function that returns each time a different random float number in the range $[a, b]$.

4.5 Object tracking

4.5.1 General method

Object tracking is an isolated part of this algorithm. It "sits" on the output of the earlier phases of the algorithm, and tries to maintain its hypothesis about what moving objects are present on the scene.

Note that in the next phase of the development this part of the algorithm will be replaced by a much larger system, which will use the results of many image processing algorithms to spot the location of a vehicle. See further sections about future work.

The method which is employed now is simple filter : generally speaking, the algorithm numbers the objects detected by the grouping phase, gives each of them a confidence rate, and tries to match each image to the previous one. For each object, if it succeeds, it keeps the same identifier and raises the confidence. New objects receive a new ID and an initial confidence. When an object "disappears", its confidence is starting to drop. As long as the confidence is still high, the object is remembered and the system keeps trying, in each image, to match this object to newly appearing objects. Once the confidence falls below some threshold, the object is removed from memory.

4.5.2 Upper and bottom confidence thresholds

The level of confidence needed for an object to start being considered as a "true" object is higher than the level under which it stops being "real". Let's look at an example with real values used in our algorithm : an object always starts with a confidence of 0.3. Each step this object is re-detected, it raises its confidence by 1.2. If it's not detected, its confidence is multiplied by 0.9.

Once an object passes 1.3, it starts being "real". It will continue to be "real" until it falls below 0.8. This eliminates oscillations that could have been caused if the upper and bottom thresholds were identical.

---

where sharper motions are found

4.6 Algorithm future work - directions and risks

4.6.1 Extended motion models

Remark : The following changes are not likely to be implemented in the framework of this thesis. The information is brought here for the academic discussion.

Currently our program uses 2D movement motion models only. That is - only 2D movement on the screen is taken into account. This model, in theory, cannot capture cases of objects that are zooming towards the camera. However in actual experiments we saw that good results can be obtained in most of the cases, except for objects which are zooming very strong.

A typical case that demonstrates the extent to which the algorithm can work with 2D models is shown in Figure 4.5. The car that comes from the left is zooming out. If we would have used zooming-enabling motion models, then this whole object should have had the same model, and could easily be grouped. In the case of 2D model this object contains a varying range of movement vectors, as seen in the image. However, with loose grouping criterion, we manage to capture this entire object together and make it one body, as demonstrated in Figure 4.6. This technique has the risk of capturing some parts of the road as well, and indeed this is one of the problems that we had to cope with. It is obvious therefore, that 2D models do not give sufficient results, and that it is desired to implement models can that model zooming.

Paper [Witt 01] is talking about a motion model made of 6 parameters - 2 for translation, 2 for zooming center and 2 for zooming extent. This information contains a lot of redundancy. In particular :

- Zoom extent is proportional in both axes, because our objects are rigid. Therefore there is a need only for one number there.
- Translation is actually zooming with an infinite zoom center.

This redundancy is hard to handle because two models, which are actually the same, could be represented differently, and will not be united when needed. We will try to shrink this model to a less redundant form. A possible option is using zooming model only. This type of model involves three numbers : the zoom extent - 1 number, the zoom center - and two numbers in polar form. Note that a pure movement is actually zooming to a point which is far from the image center. Therefore when we would want to model the movement (0,1) - a vector that moves up - we would express it as an un-zooming to the point (PI/2, INF) - a zoom center which is found straight up in the infinity. For practical cases, the infinity value will be any number above some threshold.

4.7 Uniqueness of motion models

In this section we will discuss the subject of uniqueness and try to develop the notion of one dimensional uniqueness.

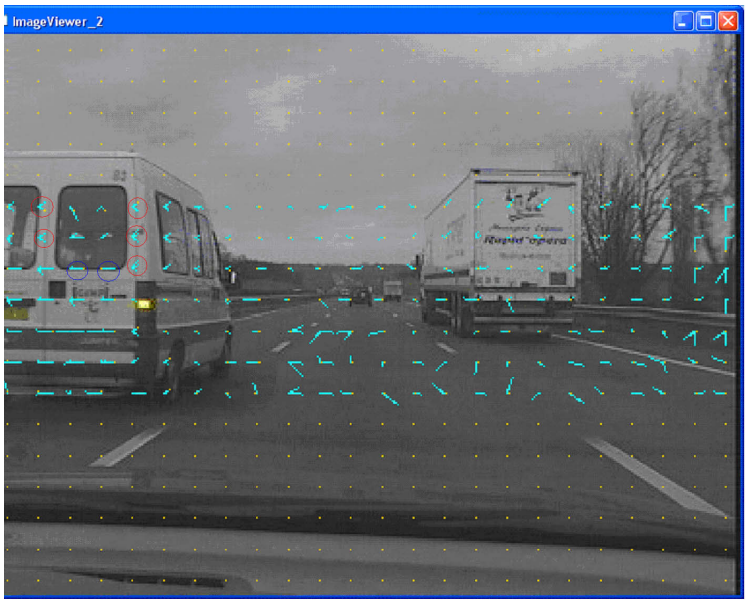


FIG. 4.5 – Edges in the car body produce multiple motion models on the same line.

FIG. 4.6 – Using smoothing, we capture the whole object even if its 2D models are varying.

4.7.1 Motion estimation for object detection and tracking

Object detection involves the segmentation of continuous regions having the same motion models. A typical scene shown in figure 4.4 shows the division of the image into two separate regions : the track entering from the right and the background.

When dealing with object detection and tracking, one is not only interested in having a low distortion, but also in having the real motion of object on the real scene. The example of the sky (where many motion models are correct at the same point) proves that low distortion is not enough. Because the absence of texture, smooth objects tend to produce motion models with low distortion, but which are not correct.

4.7.2 Simple uniqueness

To overcome this problem, we note that a smooth object contains many motion models with low distortion. These motion models can be completely different from each other. On the other hand, a motion model which exists on the real scene, will be the only one giving a low level of distortion. Therefore, the key measure of a block is the *uniqueness* of its best motion model.

Given a block B, assume that we found a motion model m^0 which minimizes the distortion on B across $\Im$. The uniqueness Φ value, in respect to B and k is defined as :

$$\Phi_k(B) = \frac{\Lambda_k(B, m^1)}{\Lambda_k(B, m^0)}$$

where m_1 minimizes the distortion on B across $\Im - m^0$. While m_0 is intuitively interpreted as the "best" motion model, the model m_1 is the "next best" one. The comparison between them actually measures how "special" is m_0 in its low distortion. Low uniqueness (towards 1) means "bad" uniqueness. A higher one means "good" uniqueness. Revisiting the previous example, a typical block in the sky will have a uniqueness smaller than 1.2, because several motion models will yield low distortion.

4.7.3 Weighted uniqueness

The use of uniqueness as defined in previous section might not be accurate, because the "best" motion model is often replaced by several similar motion models with low distortion. An object having the movement of m^{real} pixels in reality (or, to be precise, in the reflection of the reality in the image), creates for its corresponding block B several motion models $m_0 \dots m_k$ where $\mid m_i - m^{real} \mid < \epsilon$[2]. The distortion of these motion models will all be low, and as a result it often happens that the uniqueness of the best motion model is very low (close to 1). In such case the uniqueness measure doesn't form a good indicator if the movement is a real one. In order to fix that, we improve the notion and define *weighted uniqueness* $\hat{\Phi}$, which takes into account the distance between motion models :

$$\hat{\Phi}_k(B) = min_{m \in \Im} \left\{ w_m \frac{\Lambda_k(B, m)}{\Lambda_k(B, m^0)} \right\}$$

[2] we use the Euclidean distance between motion models.

where $w_m = 1 - e^{-|m-m^0|}$ is the weight of the motion model, giving low values for motion models close to m_9 and getting very close to 1 as the distance to m_0 is larger than 2 pixels. This calculation, as in simple uniqueness, calculates the ratio between the distortion of the "second best" motion model and the distortion of the best motion model. But here, the "second best" is not automatically taken to be the one having the lowest distortion across the set $\Im - m_0$, but searches for one also having a reasonable distance to m_0. A motion model having a low distance to m_0 will have a low weight and thus will not be the most dominant in the calculation of the uniqueness.

We note that for purposes of efficiency, the weight function w_m can be approximated by :

$w'_m = \{ \begin{array}{ll} 1 & \text{if } \mid m - m^0 \mid > 1.5\epsilon \\ 0 & \text{otherwise} \end{array}$

We also note that in practice, one doesn't check the distortion on the entire set $\Im$, unless full search is used. Instead, a smaller set $\Im' \subset \Im$ is checked, according to the block matching algorithm used.

Figure 4.4 is a good example. Marked blocks are ones with uniqueness value larger than 1.3. As we saw before, this choice well detects moving objects, without any false detection of movement.

4.7.4 Towards one dimensional uniqueness

A close observation of some particular cases (see figure 4.5) shows that by dropping the blocks without unique motion models we might lose important information. The blocks marked by red circles do not contain a unique motion model with low distortion, because they fall on a smooth part of an object. However, the smoothness goes only vertically, and therefore all their low-distortion motion models lie on the same line. This observation sparks the notion of *one-dimensional uniqueness*, which is a generalization of the uniqueness measure. To develop this notion, we need some further definitions.

4.7.5 Uniqueness as statistical dispersion

One can give a somewhat different definition of uniqueness. Given a block B at frame k of the video sequence, let us look at the set of all motion models $\Im$ as a set of 2D weighted data points, where for each model $m \in \Im$ its weight is $w_{B,k}(m) = e^{-\Lambda_k(B,m)/\rho}$ for some value of ρ. From this type of representation, it seems that our intuitive definition of uniqueness collides with the *weighted variance* of these data :

$$\tilde{\Phi}_k(B) = \frac{\sum_{m\in\Im} w_{B,k}(m) \mid m - \bar{m}_{B,k} \mid^2}{\sum_{m\in\Im} w_{B,k}(m)}$$

where $\bar{m}_{B,k} = \sum_{m\in\Im} m w_{B,k}(m)$ is the weighted average of the data points.

Note that $\tilde{\Phi}$ can be quickly computed as the motion models with the distortion higher than 2ρ can be omitted.

4.7.6 One dimensional uniqueness as the weighted least squares

Similar to the statistical uniqueness $\tilde{\Phi}$, we can develop the notion of one-dimensional uniqueness as a statistical measure of the same data points. However this time we will measure the dispersion of these points around a line and not a point. The measure we use is the *weighted least squares* :

$$\tilde{\Phi}_k^{1d}(B) = \frac{\sum_{m \in \Im} w_{B,k}(m) dist(m, L_{B,k})^2}{\sum_{m \in \Im} w_{B,k}(m)}$$

where $L_{B,k} = Ax + By + C$ is the line which minimizes $\tilde{\Phi}_k^{1d}$. This line can be calculated using the standard method, by taking 3 partial derivatives of $\tilde{\Phi}_k^{1d}$ with respect to A, B and C, comparing to zero and solving the resulting system of three equations with three unknowns.

4.7.7 Using the one dimensional uniqueness to enhance detection

We revisit the example shown in figure 4.5. This time, we used the newly defined $\tilde{\Phi}$. in the figure we marked the weighted average motion model for blocks with $\tilde{\Phi} < 1.3$ (Note that with $\tilde{\Phi}$ the threshold is opposite than with $\hat{\Phi}$). In addition we show the one dimensional uniqueness and mark the best regression line for $\tilde{\Phi}_k^{1d} < 1.3$.

Because of the nature of objects, many times we will find a block B_1 where $\tilde{\Phi}_k(B_1)$ is low (a "2D-unique" block), with proximity to another block B_2 having high $\tilde{\Phi}_k(B_2)$ but low $\tilde{\Phi}_k^{1d}(B_1)$ (a "1D-unique block"), and $\bar{m}_{B_1,k}$ lies or almost lies on the line $L_{B_2,k}$. This case occurs in figure 4.5. In such case, we can propagate the best motion model of B_1 to B_2.

4.8 Conclusion

In this chapter we have presented an algorithm for motion estimation. We started from an existing algorithm described by Wittebrood and de Haan [Witt 01], and improved it to meet our needs in the domain of visual detection of objects from an on-board camera. In particular, we have developed the notion of *uniqueness* as a tool to handle moving objects.

In the next chapters, we will see how the motion algorithm is taking its place near the AdaBoost based algorithms in order to create a visual detection and tracking system.

Chapitre 5

Particles filter

5.1 Introduction

In order to develop efficient intelligent-vehicle applications it is not enough to have the visual detection and motion estimation techniques described in the last two chapters. In order to track and estimate the localization and speed of objects (cars or pedestrians) one needs an efficient filter.

This problem is actually quite general : like many scientific problems, we require here an estimation of the state of a system that changes over time according to a sequence of noisy measurements taken from the system. The noisy measurements, in our case, might be the detection algorithm and movement estimation described in the previous chapters, as well as other traditional vision algorithms.

In this chapter, we will describe methods to solve this problem. We will not speak on the specific problem (detection and tracking of cars or pedestrians), but we will speak about the general problem. However, will not be too general as well : because we deal with systems being fed by a video camera, we will concentrate on the state-space approach for the modeling of dynamic systems - that is, on the discrete-time formulation of the general theoretical problem. Thus, we will develop equations and algorithms to model the evolution of the system along time, and assume that measurements are available at discrete times (which is indeed the case, since each image provides measurements).

The following is taken from [Arul 02]. The main idea in such modeling is to have a **state vector**. Such a vector contains all the relevant information needed to describe the system that we are investigating. In addition, we have a **measurement vector**. The measurement vector represents noisy observations that are related to the state vector. The measurement vector is generally, but not necessarily, of lower dimension than the state vector.

To model a system we need two models :

1 **System model** : a model describing the evolution of the state with time.

2 **Measurement model** : a model relating the noisy measurements to the state.

We will assume that these models are available in a probabilistic form. The probabilistic state-space formulation and the requirement for the updating of information on receipt of

new measurements are ideally suited for the Bayesian approach. This provides a rigorous general framework for dynamic state estimation problems.

In the Bayesian approach to dynamic state estimation, one attempts to construct the posterior probability density function (pdf) of the state based on all available information, including the set of received measurements. Since this pdf embodies all available statistical information, it may be said to be the complete solution to the estimation problem.

In principle, an optimal (with respect to any criterion) estimate of the state may be obtained from the pdf. A measure of the accuracy of the estimate may also be obtained. For many problems, an estimate is required every time that a measurement is received. In this case, a recursive filter is a convenient solution. A recursive filtering approach means that received data can be processed sequentially rather than as a batch so that it is not necessary to store the complete data set nor to reprocess existing data if a new measurement becomes available. Such a filter consists of essentially two stages : prediction and update. The prediction stage uses the system model to predict the state pdf forward from one measurement time to the next. Since the state is usually subject to unknown disturbances (modeled as random noise), prediction generally translates, deforms, and spreads the state pdf.

The update operation uses the latest measurement to modify the prediction pdf. This is achieved using Bayes theorem, which is the mechanism for updating knowledge about the target state in the light of extra information from new data.

The following is taken from [Arul 02]. We begin in Section 5.2 with a description of the nonlinear tracking problem and its optimal Bayesian solution. When certain constraints hold, this optimal solution is tractable. The Kalman filter and grid-based filter, which is described in Section 5.3, are two such solutions. Often, the optimal solution is intractable. Particles filter is described in Section 5.4.

5.2 Nonlinear Bayesian Tracking

To define the problem of tracking, consider the evolution of the state sequence $\{x_k,\ k \in \mathcal{N}\}$ of a target given by :

$$x_k = f_k(x_{k-1}, v_{k-1}) \tag{5.1}$$

where $f_k : \mathcal{R}^{N_x+N_v} \rightarrow \mathcal{R}^{N_x}$ is a possibly nonlinear function of the state x_{k-1}, $\{v_{k-1},\ k \in \mathcal{N}\}$ is a process noise sequence and N_x and N_v are dimensions of the state and process noise vectors, respectively. The objective of tracking is to recursively estimate x_k from measurements :

$$z_k = h_k(x_k, n_k) \tag{5.2}$$

where $h_k : \mathcal{R}^{N_x+N_n} \rightarrow \mathcal{R}^{N_z}$ is a possibly nonlinear function, $\{n_k,\ k \in \mathcal{N}\}$ is a measurement noise sequence and N_z and N_n are dimensions of the measurements and measurement noise vectors, respectively. In particular, we seek filtered estimates of x_k based on the set of all available measurements $z_{1:k} = \{z_i,\ i = 1 \dots k\}$ up to time k.

From a Bayesian perspective, the tracking problem is to recursively calculate some degree of belief in the state x_k at time k, taking different values, given the data $z_{1:k}$ up to

time k. Thus, it is required to construct the pdf $p(x_k\|z_{1:k})$. It is assumed that the initial pdf $p(x_0\|z_0) \equiv p(x_0)$ of the state vector, which is also known as the prior, is available (z_0 is actually an empty set of measurements). Then, in principle, the pdf $p(x_k\|z_{1:k})$ may be obtained, recursively, in two stages : prediction and update.

Suppose that the required pdf $p(x_{k-1}\|z_{1:k-1})$ at time $k-1$ is available. The prediction stage involves using the system model to obtain the prior pdf of the state at time k via the Chapman-Kolmogorov equation :

$$p(x_k\|z_{1:k-1}) = \int p(x_k\|x_{k-1})p(x_{k-1}\|z_{1:k-1})dx_{k-1} \tag{5.3}$$

In equation 5.3 we used the fact that $p(x_k|x_{k-1}z_{1:k-1}) = p(x_k\|x_{k-1})$ because equation 5.1 is a Markov process of order one. The probabilistic model of the state evolution $p(x_k\|x_{k-1})$ is defined in equation 5.1.

At time step k, a measurement z_k becomes available, and this may be used to update the prior (update stage) via Bayes' rule :

$$p(x_k\|z_{1:k}) = \frac{p(z_k\|x_k)p(x_k\|z_{1:k-1})}{p(z_k\|z_{1:k-1})} \tag{5.4}$$

where the normalizing constant

$$p(z_k\|z_{1:k-1}) = \int p(z_k\|x_k)p(x_k\|z_{1:k-1})dx_k \tag{5.5}$$

depends on the likelihood function $p(z_k\|x_k)$ defined by the measurement model 5.2 and the known statistics of n_k. In the update stage 5.4, the measurement z_k is used to modify the prior density to obtain the required posterior density of the current state. The recurrence relations 5.3 and 5.4 form the basis for the optimal Bayesian solution. This recursive propagation of the posterior density is only a conceptual solution in that in general, it cannot be determined analytically. Solutions do exist in a restrictive set of cases, including the Kalman filter and grid-based filters described in the next section.We also describe how, when the analytic solution is intractable, extended Kalman filters, approximate grid-based filters, and particle filters approximate the optimal Bayesian solution.

5.3 Optimal Algorithms

5.3.1 Kalman Filter

The Kalman filter assumes that the posterior density at every time step is Gaussian and, hence, parameterized by a mean and covariance.

If $p(x_{k-1}\|z_{1:k-1})$ is Gaussian, it can be proved that $p(x_k\|z_{1:k})$ is also Gaussian, provided that certain assumptions hold :

- v_{k-1} and n_k are drawn from Gaussian distributions of known parameters.
- $f_k(x_{k-1}, v_{k-1})$ is known and is a linear function of x_{k-1} and v_{k-1}.
- $h_k(x_k, n_k)$ is a known linear function of x_k and n_k.

That is, equation 5.1 and 5.2 can be rewritten as :

$$x_k = F_k x_{k-1} + v_{k-1} \tag{5.6}$$

$$z_k = H_k x_k + n_k \tag{5.7}$$

F_k and H_k are known matrices defining the linear functions. The covariances of v_{k-1} and n_k are, respectively, Q_{k-1} and R_k. Here, we consider the case when v_{k-1} and n_k have zero mean and are statistically independent. Note that the system and measurement matrices F_k and H_k, as well as noise parameters Q_{k-1} and R_k, are allowed to be time variant.

The Kalman filter algorithm, which was derived using 5.3 and 5.4, can then be viewed as the following recursive relationship :

$$p(x_{k-1} \| z_{1:k-1}) = \mathcal{N}(x_{k-1}; m_{k-1\|k-1}, P_{k-1\|k-1}) \tag{5.8}$$

$$p(x_k \| z_{1:k-1}) = \mathcal{N}(x_{k-1}; m_{k\|k-1}, P_{k\|k-1}) \tag{5.9}$$

$$p(x_k \| z_{1:k}) = \mathcal{N}(x_{k-1}; m_{k\|k}, P_{k\|k}) \tag{5.10}$$

where :

$$m_{k\|k-1} = F_k m_{k-1\|k-1} \tag{5.11}$$

$$P_{k\|k-1} = Q_{k-1} + F_k P_{k-1\|k-1} F_k^T \tag{5.12}$$

$$m_{k\|k} = m_{k\|k-1} + K_k(z_k - H_k m_{k\|k-1}) \tag{5.13}$$

$$P_{k\|k} = P_{k\|k-1} - K_k H_k P_{k\|k-1} \tag{5.14}$$

$$S_k = H_k P_{k\|k-1} - K_k H_k P_{k\|k-1} \tag{5.15}$$

$$K_k = P_{k\|k-1} - K_k H_k P_{k\|k-1} \tag{5.16}$$

The notation $\mathcal{N}(x; m, P)$ stands for a Gaussian density with argument x, mean m, and covariance P.

S_k is the covariance of the innovation term $z_k - H_k m_{k\|k-1}$, and K_k is the Kalman gain.

This is the optimal solution to the tracking problem if the (highly restrictive) assumptions hold. The implication is that no algorithm can ever do better than a Kalman filter in this linear Gaussian environment. It should be noted that it is possible to derive the same results using a least squares (LS) argument [Jazw 70]. All the distributions are then described by their means and covariances, and the algorithm remains unaltered, but the distributions are not constrained to be Gaussian. Assuming the means and covariances to be unbiased

and consistent, the filter then optimally derives the mean and covariance of the posterior. However, this posterior is not necessarily Gaussian, and therefore, if optimality is the ability of an algorithm to calculate the posterior, the filter is then not certain to be optimal.

5.3.2 Grid-based Methods

Grid-based methods provide the optimal recursion of the filtered density $p(x_k \| z_{1:k})$ if the state space is discrete and consists of a finite number of states. Suppose the state space at time $k-1$ consists of discrete states $x^i_{k-1},\ i = 1 \dots N_s$. For each state x^i_{k-1}, let the conditional probability of that state, given measurements up to time $k-1$ be denoted by $w^i_{k-1\|k-1}$. In other words : $Pr(x_{k-1} = x^i_k) = w^i_{k-1\|k-1}$. Then, the posterior pdf at $k-1$ can be written as :

$$p(x_{k-1} \| z_{1:k-1}) = \sum_{i=1}^{N_s} w^i_{k-1\|k-1} \delta(x_{k-1} - x^i_{k-1}) \tag{5.17}$$

where *delta* is the Dirac delta measure. Substitution of 5.17 into 5.3 and 5.4 yields the prediction and update equations, respectively :

$$p(x_k \| z_{1:k-1}) = \sum_{i=1}^{N_s} w^i_{k\|k-1} \delta(x_k - x^i_k) \tag{5.18}$$

$$p(x_k \| z_{1:k}) = \sum_{i=1}^{N_s} w^i_{k\|k} \delta(x_k - x^i_k) \tag{5.19}$$

Note that $w^i_{k\|k-1}$ can be calculated using the known probability (actually, the model evolvement) $p(x^i_k \| x^j_{k-1})$:

$$w^i_{k\|k-1} = \sum_{j=1}^{N_s} w^j_{k-1\|k-1} p(x^i_k \| x^j_{k-1}) \tag{5.20}$$

and that $w^i_{k\|k}$ can be calculated using the known probability (actually, the likelihood function) $p(z_k \| x^i_k)$:

$$w^i_{k\|k} = \frac{w^i_{k\|k-1} p(z_k \| x^i_k)}{\sum_{j=1}^{N_s} w^j_{k\|k-1} p(z_k \| x^i_k)} \tag{5.21}$$

We assume that the likelihood function and the model evolution are known (otherwise, the problem is undefined..) and that these probabilities can appear in any form, as long that they can be calculated.

Again, this is the optimal solution if the assumptions made hold.

5.4 Particles filter

5.4.1 Sequential Importance Sampling (SIS) Algorithm

The sequential importance sampling (SIS) algorithm is a Monte Carlo (MC) method that forms the basis for most sequential MC filters developed over the past decades ; see [Douc 02] (in the chapter "An introduction to sequential Monte Carlo methods"), or [Douc 01]. This sequential MC (SMC) approach is known variously as bootstrap filtering [Gord 93], the condensation algorithm [MacC 99], particle filtering [Carp 99], interacting particle approximations [Cris 99], [Mora 96], and survival of the fittest [Kana 95]. It is a technique for implementing a recursive Bayesian filter by MC simulations. The key idea is to represent the required posterior density function by a set of random samples with associated weights and to compute estimates based on these samples and weights. As the number of samples becomes very large, this MC characterization becomes an equivalent representation to the usual functional description of the posterior pdf, and the SIS filter approaches the optimal Bayesian estimate. In order to develop the details of the algorithm, let $\{x^i_{0:k}, w^i_k\}^{N_s}_i$ denote a random measure that characterizes the posterior pdf $p(x_{0:k}\|z_{1:k})$, where $\{x^i_{0:k}, i = 0 \dots N_S\}$ is a set of support points with associated weights $\{w^i_k, i = 0 \dots N_S\}$ and $x_{0:k} = \{x_j, j = 0 \dots k\}$ is the set of all states up to time k. The weights are normalized such that $\Sigma_i w^i_k = 1$. Then, the posterior density at k can be approximated as :

$$p(X_{0:k}|Z_{1:k}) \approx w^i \delta(x - x^i) \tag{5.22}$$

where :

$$w^i \propto \frac{\pi(x^i)}{q(x^i)} \tag{5.23}$$

is the normalized weight of the ith particle.

Therefore, if the samples $X^i_{0:k}$ were drawn from an importance density $q(x_{0:k}\|z_{1:k})$, then the weights in 5.22 are defined by 5.23 to be :

$$w^i \propto \frac{p(x_{0:k}\|z_{1:k})}{q(x_{0:k}\|z_{1:k})} \tag{5.24}$$

Returning to the sequential case, at each iteration, one could have samples constituting an approximation to $p(x_{0:k-1}\|z_{1:k-1})$ and want to approximate $p(x_{0:k}\|z_{1:k})$ with a new set of samples. If the importance density is chosen such that

$$q(x_{0:k}\|z_{1:k}) = q(x_1|x_{0:k}, z_{1:k})q(x_{0:k-1}, z_{1:k-1}) \tag{5.25}$$

then one can obtain samples $x^i_{0:k} \propto q(x_{0:k}\|z_{1:k})$ by augmenting each of the existing samples $x^i_{0:k-1} \propto q(x_{0:k-1}\|z_{1:k-1})$ with the new state $x^i_k \propto q(x_k\|x_{0:k-1}\|z_{1:k})$. To derive the weight update equation $p(x_{0:k}\|z_{1:k})$, it is first expressed in terms of $p(x_{0:k-1}, z_{1:k-1})$, $p(z_k, x_k)$ and $p(x_k, x_{k-1})$. A series of computations can show that the new weights are calculated by the old weights according to :

SIS Particle Filter
$[\{X_k^i, w_k^i\}_{i=1}^{N_0}] = SIS[\{X_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_0}, z_k]$
- FOR i = 1 : N_s
  - Draw $x_k^i \sim q(X_k \| X_{k-1}^i, z_k)$
  - Assign the particle a weight, w_k^i, according to 5.27
- END FOR

TAB. 5.1 – The SIS particle filter algorithm.

$$w_k^i \propto \frac{p(z_k, x_k)p(x_k, x_{k-1})}{q(x_k^i \| x_{0:k-1}^i, z_{1:k})} \tag{5.26}$$

Furthermore, if $q(x_k \| x_{0:k-1}, z_{1:k}) = q(x_k \| x_{k-1}, z_k)$, then the importance density becomes only dependent on x_{k-1} and z_k. This is particularly useful in the common case when only a filtered estimate of $p(x_k \| z_{1:k})$ is required at each time step. From this point on, we will assume such a case, except when explicitly stated otherwise. In such scenarios, one can discard the path and history of observations $z_{1:k-1}$. The modified weight is then

$$w_k^i \propto \frac{p(z_k, x_k)p(x_k, x_{k-1})}{q(x_k^i \| x_{k-1}^i, z_k)} \tag{5.27}$$

and the posterior filtered density $p(x_k \| z_{1:k})$ can be approximated as :

$$p(x_k \| z_{1:k} \approx \sum_{i=1}^{N_S} w_k^i \delta(x_k - x_k^i) \tag{5.28}$$

where the weights are defined in (5.27). It can be shown that as $N_S \to \infty$, the approximation (5.28) approaches the true posterior density $p(x_k \| z_{1:k})$. The SIS algorithm thus consists of recursive propagation of the weights and support points as each measurement is received sequentially. A pseudo-code description of this algorithm is given by algorithm 5.1.

5.4.2 Resampling and the generic particles filter

A common problem with the SIS particle filter is the degeneracy phenomenon, where after a few iterations, all but one particle will have negligible weight. It has been shown [Douc 01] that the variance of the importance weights can only increase over time, and thus, it is impossible to avoid the degeneracy phenomenon. This degeneracy implies that a large computational effort is devoted to updating particles whose contribution to the approximation to $p(X_k | Z_{1:k})$ is almost zero. A suitable measure of degeneracy of the algorithm is the effective sample size N_{EFF} introduced in [Berg 99] and [Liu 98], and defined as

$$N_{EFF} = \frac{N_S}{1 + var(w_k^{*i})} \tag{5.29}$$

where $w_k^{*i} = p(x_k^i|z_{1:k})/q(x_k^i|x_{k-1}^i, z_k)$ is referred to as the "true weight". This cannot be evaluated exactly, but an estimate $\widehat{N_{EFF}}$ of N_{EFF} can be obtained by :

$$\widehat{N_{EFF}} = \frac{1}{\sum_{i=1}^{N_S}(w_k^i)^2} \tag{5.30}$$

where w_k^i is the normalized weight obtained using (5.26). Notice that $N_{EFF} \leq N_S$, and small N_{EFF} indicates severe degeneracy. Clearly, the degeneracy problem is an undesirable effect in particle filters. The brute force approach to reduce its effect is to use a very large N_S. This is often impractical ; therefore, we rely on another method called resampling.

The effects of degeneracy can be reduced to use resampling whenever a significant degeneracy is observed (i.e., when N_{eff} falls below some threshold N_T). The basic idea of resampling is to eliminate particles that have small weights and to concentrate on particles with large weights. The resampling step involves generating a new set $\{X_k^i\}_{i=1}^{N_s}$ by resampling (with replacement) N_S times from an approximate discrete representation of $p(x_k|z_{1:k})$. The resulting sample is in fact an i.i.d.[1] sample from the discrete density ; therefore, the weights are now reset to $w_k^i = \frac{1}{N_S}$. It is possible to implement this resampling procedure in $O(N_S)$ operations by sampling N_S ordered uniforms using an algorithm based on order statistics [Carp 99], [Ripl 87]. Note that other efficient (in terms of reduced MC variation) resampling schemes, such as stratified sampling and residual sampling [Liu 98], may be applied as alternatives to this algorithm.

A generic particle filter is then as described by table 5.2. Although the resampling step reduces the effects of the degeneracy problem, it introduces other practical problems. First, it limits the opportunity to parallelize since all the particles must be combined. Second, the particles that have high weights w_k^i are statistically selected many times. This leads to a loss of diversity among the particles as the resultant sample will contain many repeated points. This problem, which is known as sample impoverishment, is severe in the case of small process noise. In fact, for the case of very small process noise, all particles will collapse to a single point within a few iterations. Third, since the diversity of the paths of the particles is reduced, any smoothed estimates based on the particles' paths degenerate. Schemes exist to counteract this effect. One approach considers the states for the particles to be predetermined by the forward filter and then obtains the smoothed estimates by recalculating the particles' weights via a recursion from the final to the first time step [Gods 00]. Another approach is to use MCMC [Carl 92].

5.4.3 Sequential Importance Resampling (SIR) Algorithm

The SIR filter proposed in [Gord 93] is an MC method that can be applied to recursive Bayesian filtering problems. The assumptions required to use the SIR filter are very weak.

[1] independent and identically distributed

Generic Particle Filter
$[\{X_k^i, w_k^i\}_{i=1}^{N_0}] = PF[\{X_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_0}, z_k]$
- FOR i = 1 : N_s
  - Draw $x_k^i \sim q(X_k \| X_{k-1}^i, z_k)$
  - Assign the particle a weight, w_k^i, according to 5.27
- END FOR
- Calculate total weight : t=SUM$[\{w_k^i\}_{i=1}^{N_0}]$
- FOR i = 1 : N_s
  - Normalize : $w_k^i = t^{-1} w_k^i$
- END FOR
- Calculate $\hat{N_{eff}}$ using 5.30
- IF $\hat{N_{eff}} < N_T$
  - Resample
- END IF

TAB. 5.2 – The generic particle filter algorithm.

The state dynamics and measurement functions $f_k(\cdot : \cdot)$ and $h_k(\cdot : \cdot)$ in (5.1) and (5.2), respectively, need to be known, and it is required to be able to sample realizations from the process noise distribution of v_{k-1} and from the prior. Finally, the likelihood function $p(z_k : x_k)$ needs to be available for pointwise evaluation (at least up to proportionality). The SIR algorithm can be easily derived from the SIS algorithm by an appropriate choice of i) the importance density, where $q(x_k|x_{k-1}^i, z_{1:k})$ is chosen to be the prior density $p(x_k|x_{k-1}^i)$, and ii) the resampling step, which is to be applied at every time index. The above choice of importance density implies that we need samples from $p(x_k|x_{k-1}^i)$. A sample $x_k^i \sim p(x_k|x_{k-1}^i)$ can be generated by first generating a process noise sample $v_{k-1}^i \sim p_v(V_{k-1})$ and setting $x_k^i = f_k(x_{k-1}^i, v_{k-1}^i)$, where $p_v(\cdot)$ is the pdf of v_{k-1}. For this particular choice of importance density, it is evident that the weights are given by :

$$w_k^i \propto w_{k-1}^i p(z_k|x_k^i) \tag{5.31}$$

However, noting that resampling is applied at every time index, we have $w_k^i = 1/N \forall i$; therefore

$$w_k^i \propto p(z_k|x_k^i) \tag{5.32}$$

The weights given by the proportionality in (5.32) are normalized before the resampling stage. An iteration of the algorithm is then described by Algorithm 5.3. As the importance sampling density for the SIR filter is independent of measurement z_k, the state space is explored without any knowledge of the observations. Therefore, this filter can be inefficient and is sensitive to outliers. Furthermore, as resampling is applied at every iteration, this can

SIR Particle Filter
$[\{X_k^i, w_k^i\}_{i=1}^{N_0}] = SIR[\{X_{k-1}^i, w_{k-1}^i\}_{i=1}^{N_0}, z_k]$
- FOR i = 1 : N_s
  - Draw $x_k^i \sim q(X_k \| X_{k-1}^i, z_k)$
  - Assign the particle a weight, w_k^i, according to 5.27
- END FOR
- Calculate total weight : t=SUM$[\{w_k^i\}_{i=1}^{N_0}]$
- FOR i = 1 : N_s
  - Normalize : $w_k^i = t^{-1} w_k^i$
- END FOR
  - Resample

TAB. 5.3 – The SIR particle filter algorithm.

result in rapid loss of diversity in particles. However, the SIR method has the advantage that the importance weights are easily evaluated and that the importance density can be easily sampled.

5.5 Conclusion

In this chapter, we have seen the need for filtering for the purpose of tracking. Towards this goal, we've passed over different methods of filtering. We have seen some optimal filters that work perfectly when the signal is linear and gaussian. Then we've presented the particle filter and exaplained why it is suitable for visual detection in-vehicle system. We have seen several types of this filter.

Chapitre 6

Application I : Stop&Go system

This chapter describes a close range ACC (so called "Stop&Go") application. This type of application is designed to be installed on a moving vehicle equipped with a single frontal camera and a lidar. The goal of the system is to automatically control the car's speed in order to keep a constant distance from the car in front. The name of the system ("Stop&Go") comes from the situation where it's the most useful - heavy traffic, where the driver has to stop and go every few seconds to follow the car in front. The system identifies the car in front, estimating the distance to it and control the car speed automatically to preserve a fix distance.

The system is activated manually by the driver. Once activated, the system searches for a target to lock on. An important feature of the system is its ability to automatically deactivate itself when it feels that it cannot reliably tell where is the car in front, or when it feels that some other obstacle has entered the space between the host vehicle and the car we follow. Such a deactivation could take place immediately after activation or after a long period of operation.

The application employs a tracking engine for detection and tracking of target vehicles in front of the host vehicle. The output of this tracking engine is a list of targets. The application selects one target (the one in front of the host vehicle) and outputs activation/deactivation command and a desired car speed. This data is transferred to a control unit which is controlling the hardware in the car (gas, brakes, etc.).

6.1 The detection and tracking framework

6.1.1 Requirements for the application

Coming to design this detection and tracking system, we should analyze the input and output of the system. The main input of the system - the images arriving from the camera - can be analyzed by several observations. Each one of these observations detects different features of the image - motion, dark areas, symmetries, vertical edges etc. The output, on the other hand, should be a list of targets, with accurate spatial position of each target, plus a measure of this accuracy (confidence value). The question, therefore, is how to go from the input observations to the output.

The problem is difficult :

- Image processing is a non-linear and non-gaussian observation process.
- Projection in an image is also non-linear and non-gaussian.
- Simple Kalman filter (typical algorithm for model inversion) is out of focus here. Extended Kalman filter might operate poorly (it is not known what model to provide).

6.1.2 Previous approaches

Bayesian Networks

Bayesian Networks-based data fusion was used in a previous project (FADE 1, see [Steu 02]), and provided good results. Unfortunately, this method exhibits severe limitations, as discussed below.

Advantages :

- This is a diagnosis approach : enables the detection of failing algorithms (especially image processing algorithms), through the computation of correlations between proposals made up from several sources.
- Observation is completely independent of the ground plane constraint, since we use a model of the width of the target for retro-projection (typically 1.75m).
- Accuracy of localization is high, since we combine many different "sensor outputs" from many different algorithms, thus yielding in a typical 10% distance accuracy.

Drawbacks :

- One target = one hypothesis (one state space position). No real state space exploration.
- Bayesian nets don't allow to distinguish between confidence and accuracy of localization of the target. When a hypothesis gets a low score, it is difficult to state whether the target is badly located (inaccuracy of detection) or the target simply doesn't exist or doesn't fit the model (uncertainty). It's very difficult to set up a "split" strategy from the results provided by the Bayesian net.
- The major drawback is that the actual setup can only use image processing algorithms that provide vertical features (in order to combine inputs). This is good for edge detection, lights, shadow, symmetry detection but how to integrate motion segmentation into that framework ?
- The fusion algorithm is difficult to tune. You have to attribute a priori belief to each source, which was found to be tricky.
- Bayesian net computation is heavy, so it's generally necessary to get back to approximations of Bayesian nets (which were done in FADE 1), which to some extent tend to deteriorate the results.

Problems :

- It is unclear how to integrate motion segmentation into such a framework.
- It is not possible to cope with different kinds of targets, like motorcycles and trucks.

6.1.3 Particle filtering

Particle filtering, as described in chapter 5, is a useful method to track targets in non-linear, non-gaussian environment, as is our system.

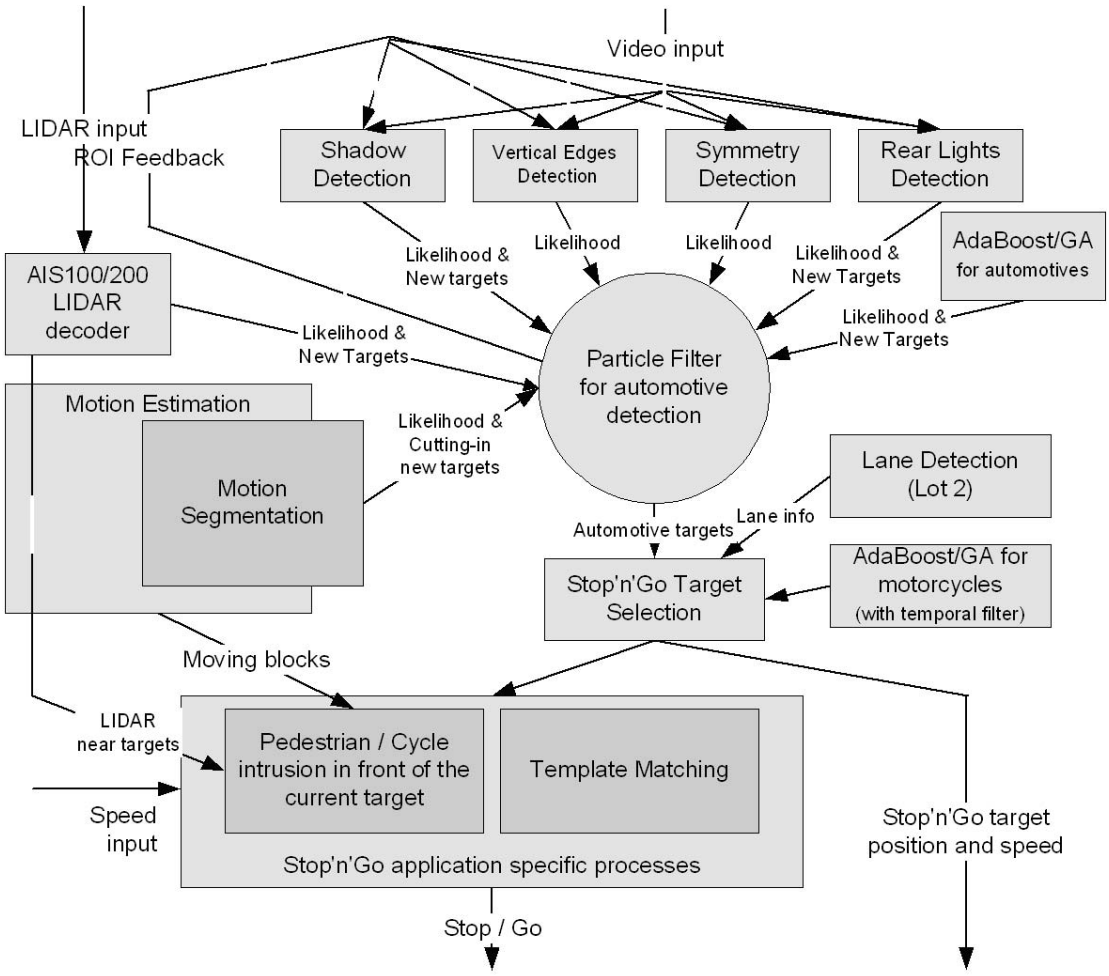


FIG. 6.1 – The fusion process.

FIG. 6.2 – The resampling process.

$\{x_k^i, w_k^i\}_{i=1}^N \leftarrow SIR(\{x_{k-1}^i, w_{k-1}^i\}_{i=1}^N, z_k)$

1. For $i = 1 \ldots N$, do :

   1.1 Draw $x_k^i \cong p(x_k \mid x_{k-1}^i)$

   1.2 Calculate $w_i = p(z_k \mid x_{k-1}^i)$

2. Send $\{x_k^i, w_k^i\}_{i=1}^N$ to output stage

3. For all $i = 1 \ldots N$, Normalize $w_k^i \leftarrow \frac{w_k^i}{t}$

Where $t = \sum_{i=1}^{N} w_k^i$

4. $\{x_k^i, w_k^i\}_{i=1}^N \leftarrow RESAMPLE(\{x_k^i, w_k^i\}_{i=1}^N)$

TAB. 6.1 – The SIR particles filter algorithm.

Figure 6.1 shows a diagram representing how we use particle filtering in an application using several image processing algorithms. The particle filter is used to make the fusion between all these algorithms, which individually are not sophisticated enough to yield a good result, but can be combined together to provide a robust system.

Our chosen version of particles filter is the SIR filter, because of its simplicity. The algorithm is given in table 6.1. The resampling algorithm appears in table 6.2. A visual explanation of the resampling process appears in figure 6.2.

We will use hereinafter the notation of *steps*. Step k of the execution corresponds to the status of the application after processing input image number k. Thus, at the beginning of the execution $k = 0$.

The system is maintaining, at each moment, a set of targets. At $k = 0$, this set is empty. New targets are initialized by several algorithms in a way that will be described later.

At step k when $k > 0$, some targets have been initialized and the targets set is possibly not empty. For each existing target we maintain a *probability density function* (PDF) $p(x_k \mid z_{1-k})$ of the state x_k of the target, according to all the observations $z_1, \ldots, z_k$ received at steps 1 to k. We use the notation z_{1-k} for these observations.

$\{x_k^i, w_k^i\}_{i=1}^N \leftarrow RESAMPLE(\{x_k^i, w_k^i\}_{i=1}^N)$

1. Initialize the CDF (cumulative density function) to $c_l = 0$

1. For $i = 2 \ldots N$, do :

  1.1 $c_l \leftarrow c_{l-1} + w_k^i$

2. Assign $i \leftarrow 1$.

3. For $j = 1 \ldots N$, do :

  3.2 While $\frac{j-1}{N} > c_i$, do $i \leftarrow i + 1$

  3.3 Assign particle's sample $x_k^j \leftarrow x_k^i$

  3.4 Assign particle's weight $w_k^j \leftarrow \frac{1}{N}$

TAB. 6.2 – The resampling algorithm.

The PDF is represented using a set of N random samples[1] with associated weights $\{x_k^i, w_k^i\}_{i=1}^N$, called *particles*Ėach indexParticleexisting target in the system, therefore, has a set of particles and each particle is a *weighted hypothesis* about that target.

6.1.4 The state space

In our particular case, the state x_k^i is a triple $\langle X, Y, T\rangle$ representing a box-like vehicle whose bottom rear edge is centered around the ground point $\langle X, Y, 0\rangle$. The parameter T can have one of the three values $\{MOTORCYCLE, CAR, TRUCK\}$, corresponding to motorcycle, car or truck.

The vehicle dimensions are determined by T using table 6.13. Vehicle is assumed to be aligned with the axes system (i.e. $\theta = 0, \phi = 0, \psi = 0$), as well as the ground, which is supposed to be flat in the area of interest.

The state space was deliberately designed to be of low dimension. Theoretically, it was ideal to include in the state space also the width and length of the vehicle, its angle and may be also its speed and acceleration. However, choosing a high-dimensional state space causes lower convergence, or alternatively a need for a huge number of particles (which in turn causes a slow operation).

Therefore, we chose to keep only $\langle x, y\rangle$ in the particle state space, plus a discrete value T of the type of the vehicle. This makes the state space easily observable with a top view of the situation, and thus eases the understanding and tuning of the system.

[1] in our implementation, $N = 100$

FIG. 6.3 – A typical vehicle target, frontal image (left) and its particles on a bird's eye view (right).

The discrete value T saves us the need to include the width in the state space since by determining the type of the vehicle we can roughly know its width.

Figure 6.3 shows a typical traffic vehicle target and its particles.

6.1.5 How algorithms are combined

When speaking about image processing algorithms in this context, we are referring to a type of algorithms which is being ran one time per image of the video sequence. Typical algorithm will get the input image. In addition, it will typically get some bounding boxes that specify where we suspect there are targets. For some algorithms, this information is crucial because they are target-specific. For others, this information is helping to perform faster, because it allows them to perform some processing only on the expected area of these targets.

After processing the input image, algorithms generally do one or both of the following operations : initialize new targets and support/decline hypotheses about the location of already-discovered targets.

Initialization of targets

To create new functions, an algorithm should give us as output a list of new targets, and for each target provide a list of theses where it believes the target is found. For example, when the motion segmentation algorithm spots a movement in the scene, and this movement does not seem to correspond with an existing target, it creates a new target. However, it does not know exactly where the target is ; it can be close or far, a little bit to the right or to the left. It therefore produces a list of theses (embodied as particles in our particle filtering system) about where the target is. In section 6.2 we explain how the different algorithms are initializing targets.

The likelihood functions

To support or decline existing targets, we use the mechanism of the *likelihood function*. Each one of the 5 participating algorithms provides, *after it has executed its initial prepro-*

FIG. 6.4 – The likelihood function. On the left, a frontal view of a typical situation. On the right, a birds-eye view of the particles (the left "cloud" of each group) and the likelihood functions of 5 algorithms (the 5 right clouds in each group)

cessing stage on the input image, a likelihood function $p(z_k \mid x_k) \in [0, 1]$, where z_k is the observation of the algorithm and x_k is the state of the target.

A visualization of typical likelihood function is shown in figure 6.4.

6.1.6 Advantages of using particle filtering

Compared to the Bayesian nets approach, using particle filtering has several advantages :

- One target = n hypotheses. This is the basis for the algorithm, and it makes fusion and splitting between targets more natural. Accuracy of localization should be high, due to the high number of particles. Averaging these positions (according to their weight) yields the global target position output.
- Visual results : eases the tuning. Tuning is done through likelihood functions, which can be of any sort (including non-linear functions) and thus are very adapted to image processing algorithms.
- Allows to distinguish between confidence (weights of particles) and inaccuracy (spatial distribution of particles).
- Allows the integration of many different algorithms (of any kind, providing any kind of input). Makes possible the integration of horizontal edge detection and the use of different models than just the width of the car (i.e. height of lights at short distance, by using the hypothesis of plane road, which is accurate at very low distance).

6.1.7 Confidence of targets

Each particle belonging to a target carries a weight, which states the probability of its hypothesis. The sum of the weights for each target is 1, for convenience of the algorithm.

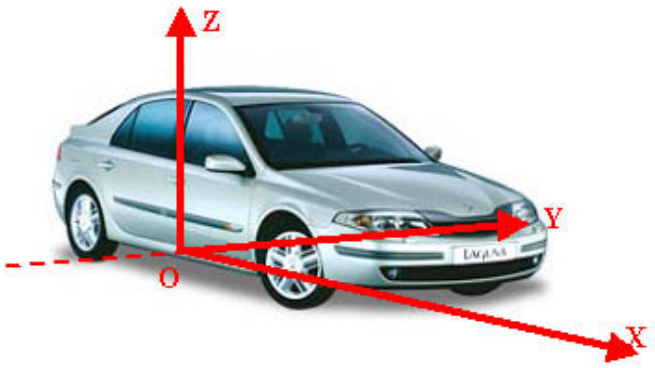


FIG. 6.5 – The coordination system.

However, we would like to have a number indicating the probability that the target exists at all. For this we maintain for each target a confidence value. The confidence is changing over the execution according to several parameters :

- The initial confidence value for a new target.
- The maximal confidence value.
- The confidence value below which a target is deleted.
- The confidence value above which a target is said to be *validated* for the purpose of the Stop&Go application.

In our implementation we used 1,1000000,0.2 and 1000 for the 4 values above.

6.2 Image processing algorithms

6.2.1 Introduction

Conventions

When we speak about a location of a vehicle on the world, we will specify a pair of the form <x,y>. Unless stated otherwise, we denote by this a vehicle which is located on the ground and its rear, bottom, middle point is at point <x,y,0>, as drawn in Figure 6.5

Projection and retro-projection

Image processing algorithms work on the screen ; real targets exist in the real world. These different coordinates need converting to both directions. In our program, we assume specific parameters of the camera, with which the video sequence was taken. Using the values of these parameters, we can provide a projection of a point <x,y,z> to a screen point <u,v> and vice versa. Note that throughout this chapter, when we say that we project a vehicle located at <x,y> to the screen, we mean that we do the following sequence of operations :

- Produce the 3D box of the vehicle, which is located at <x,y> (as defined above).
- Project all the 8 points of this box.

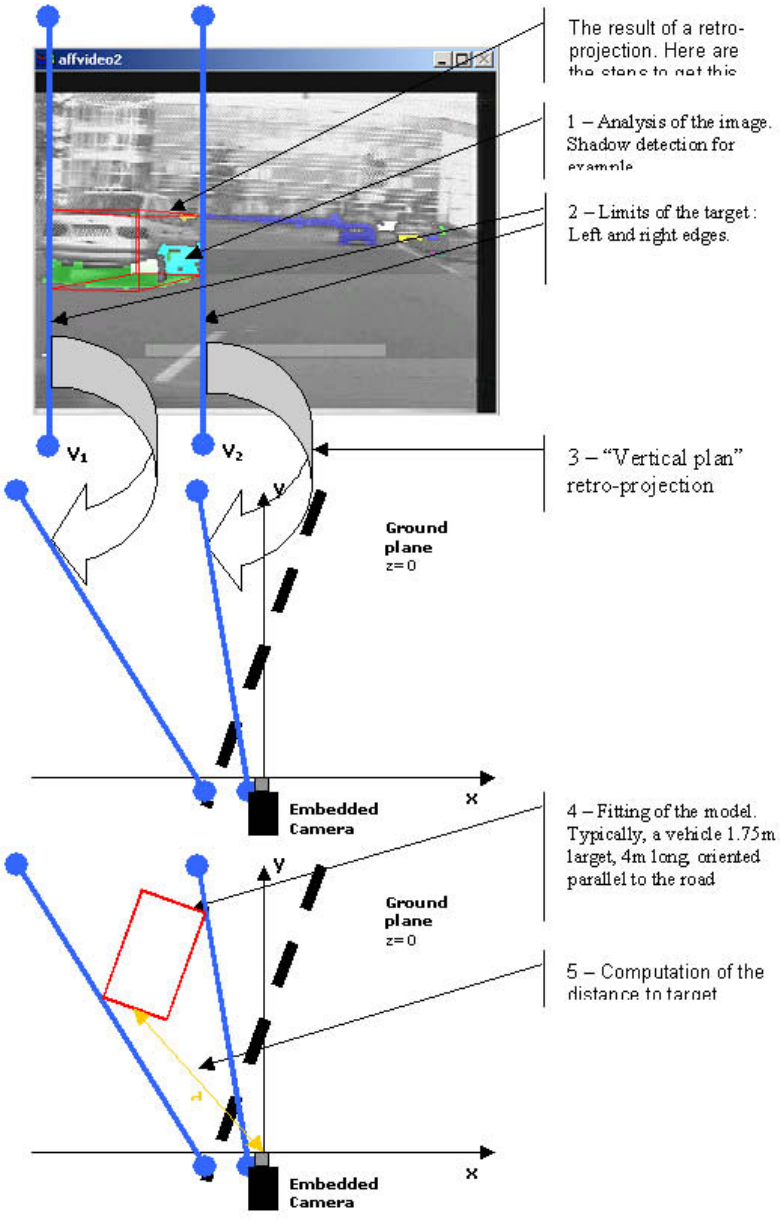


FIG. 6.6 – The retroprojection method

FIG. 6.7 – Shadows corresponding to cars

– Calculate on the screen the 3 following rectangles : the rectangle R^{rear} which is the projection of the rear face of the vehicle ("the rear projected rectangle"), the rectangle R^{front} which is the projection of the front face of the vehicle ("the front projected rectangle"), and the rectangle R^{global} which is the union of the two first rectangles ("the global projected rectangle").

When we say "the projection rectangle", without specifying which of the 3 rectangles we mean, then we mean the global rectangle.

Figure 6.6 shows the projection and retroprojection method.

6.2.2 Vehicle shadow detection

Shadow detection is based on the thresholding of images to find areas of darkness on the road. These areas are often shadows made by vehicles (see figure 6.7). The results are used both to generate new targets and to support/decline the existence of already discovered targets.

One of the parameters of the algorithm is a rectangle which is likely to fall entirely on the road, as demonstrated on Figure 6.8 (top right). As seen in the figure, this rectangle should be selected according to camera parameters. If, in the course of the execution, the algorithm suspects (we will later see how) that the rectangle is *not* falling on the road, it deactivates automatically and no analysis is done. Otherwise, it computes the histogram of pixel values on this rectangle. A typical histogram, as the one shown in figure 6.8 (top left), will consist of a narrow value, which is the pixel value of the color of the road.

The histogram is used to calculate the variance of the pixel values in the rectangle. If the variance is larger than a threshold[2], the algorithm concludes that the rectangle is not entirely on the road. See figure 6.14 for an example.

The algorithm thresholds the entire image with a threshold value equal to 90% of the peak of the histogram. The result is that all objects which are darker than the road are appearing in the thresholded image. On the area of the road, these are typically shadows of

[2]in our implementation, this threshold is equal to 60.

FIG. 6.8 – The shadow detection algorithm. The histogram (top left) is calculated on the road rectangle (top right). The image is thresholded (bottom left) and later eroded (bottom right).

obstacles on the road, and this is the idea behind this algorithm. To smooth noise, we apply a simple temporal filter on the threshold value. If we consider that in each step k we obtain a threshold value of m, we maintain at all times a smoothed threshold value t_k computed as follows :

$t_0 = 0$

$t_k = m * \gamma + t_{k-1} * (1 - \gamma)$

This filter uses a factor γ to blend the newest result with previous ones. Typical useful value for γ is 0.9.

After thresholding, the image is full of small parts, which are not valuable for analysis of shadows on the road. We use Erosion with a simple 3x3 element to remove these small parts. The two images at the bottom of figure 6.8 show the result of this operation.

Next, we perform an analysis of the blobs in the thresholded image, and use the parameters of the camera and our knowledge of the approximate width of cars, trucks and motorcycles to determine if a blob really represents one of these objects. The blob shown in figure 6.15 for example, cannot represent a car, a truck and a motorcycle. According to the parameters of the camera, we can tell that this blob represents an object of width of several dozens of meters, and this cannot be a vehicle. We filter, therefore, a large number of blobs before interpreting them into target vehicles.

The preprocessing of the image in this algorithm includes therefore the histogram, the thresholding and the blob analysis. The output of the preprocessing is a list of blobs. We will see below how this list is used to evaluate existing targets (by the likelihood function) and to create new targets.

A summary of the preprocessing of this algorithm is given in table 6.3.

The likelihood function

The input of the likelihood function, as explained in subsection 6.1.5, is a single particle which represents a single hypothesis for the location and type of a specific target. As explained in subsection 6.1.4, the hypothesis is a triple $\langle X, Y, T \rangle$, with $\langle X, Y \rangle$ being the location, in world coordinates, and $T \in \{MOTORCYCLE, CAR, TRUCK\}$ being the type. The likelihood function will try to evaluate the probability that this hypothesis is correct[3], using the list of blobs produced in the preprocessing phase.

The entire likelihood function is given in table 6.4

There are limits, beyond which information is beginning to become fuzzy. We have found that limits as the ones appearing in figure 6.18 are the most adequate. In terms of table 6.4, we have $\hat{X} = 60, \hat{Y} = 15, \bar{X} = 10, \bar{Y} = 7$. Beyond these limits, we do not try to estimate the probability. Rather than that, we return a probability of 0.5 to represent the fact that in such a distance, this algorithm cannot donate any meaningful information. This is shown in steps 2-4 of the function.

Next thing to do is to determine the width of the object we are trying to match to the blobs. We calculate, according to table 6.13, the desired width. Then, we project (step 5

[3]When saying "correct" we have to remember that we are speaking about a discrete sample of a continuous density function and therefore we are actually evaluating the environment of this hypothesis.

Input : an image I_k, coming from the Y plane of the input YUV image at step k of the execution, and a value t_{k-1} from the previous step (if $k = 0$, we consider $t_{k-1} = 0$).

Parameters : a rectangle R^{road} which normally falls on the road, a factor γ for the temporal filter and a variance threshold δ.

Output : a list of blobs $b_1 \dots b_n$.

The algorithm :

• Calculate the histogram $v_0, v_1 \dots v_{255}$ of the pixel values in R^{road} (v_i is the number of times the pixel value i appears in this rectangle). From the histogram, calculate the mean $\bar{x} = \frac{1}{N}\sum_{i=0}^{255} i * v_i$, the variance $\sigma^2 = \frac{1}{N}\sum_{i=0}^{255} v_i * (i - \bar{x})^2$ and one of the maximal values $m \in \{v \mid v \geq v_i \forall i\}$. N is the surface (in pixels) of the rectangle R^{road}.

• If $\sigma^2 > \delta$, deactivate the algorithm and exit.

• Calculate $t_k = m * \gamma + t_{k-1} * (1 - \gamma)$

• Threshold the image I_k with the threshold value $0.9 * t_k$.

• Erode the image with the following kernel e^{2x3} to remove noise :

$$e^{2x3} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

• Analyze the resulting image and obtain the output list of connected bodies, or blobs, $b_1 \dots b_n$. Each blob is represented by its bounding rectangle on the image.

TAB. 6.3 – The shadow detection preprocessing phase.

of the function) the vehicle (whose bottom rear center is located at $\langle X, Y \rangle$) on the screen, according to the camera parameters.

The projection is three-dimensional - that is, it is a 8-point box, as explained in subsection 6.2.1. We know that shadows can be created by the rear of the vehicle or by the entire vehicle (see figure 6.16). Therefore, we will try to match blobs to both the rear projected rectangle and to the global projected rectangle.

A good test to robustify this algorithm is the *two-bases* check. Without this check, there are many blobs which match to the particle's hypothesis projection, like the ones shown in figure 6.17. These blobs have the common feature that they are not horizontal. The two-bases check verifies that the blob intersects with the left and right bottom rectangles of the vehicle, as demonstrated in figure 6.19. Intersection should occupy at least 10% of the rectangle area ; one or two pixels - which can be caused by a noisy image - are not enough. This test can be seen in step 6 of the function, with $\lambda = 0.1$.

Note, that it is not always the case that a single vehicle produces a single shadow. Often, shadows are not strong and are divided into several pieces. This can be seen in figure 6.21. Therefore, to achieve better reliability, we will allow the matching of different blobs to the right and to the left sides of the vehicle.

Now we go over all the blobs, and try to match blobs with the projected vehicle. We do not take all the blobs, but only ones complying with certain conditions. First (step 7.1 of the function) we omit very small blobs, because they are too small to conclude the existence of a target. In terms of table 6.4, we use $\hat{w} = 6, \hat{h} = 2$.

A special treatment is given to hypothesis (particles) where the target vehicle is not entirely located inside the image (step 7.2). Such a case, like the one shown in image 6.22, should be treated differently because we look for only one shadow. The hidden side of the vehicle should not be matched with a shadow. In such case, we use *only* blobs which are touching the image borders to match the visible side of the vehicle.

Third, we check vertical consistency. We compare the bottom of the blob with the bottom of the global projected rectangle, and allow freedom of $V = 12$ pixels (see step 7.3).

After the matching operation we have :

1 a blob minimizing the distance between the left of the blob and the left of the global projected rectangle of the vehicle.

2 a blob minimizing the distance between the right of the blob and the right of the global projected rectangle of the vehicle.

3 a blob minimizing the distance between the left of the blob and the left of the rear projected rectangle of the vehicle.

4 a blob minimizing the distance between the right of the blob and the right of the rear projected rectangle of the vehicle.

As seen in the table (steps 8, 9 and 10), we calculate a probability between 0 and 1 according to the best left distance (the minimum between the blobs in 1 and 3) and the best right distance (2 and 4). For vehicles which fall outside of the image, we use only the distance which relates to the visible side.

Input : a particle containing a hypothesis $\langle X, Y, T\rangle$ about an existing target, where $\langle X, Y\rangle$ is the position of this target and $T \in \{MOTORCYCLE, CAR, TRUCK\}$ is its type.

Parameters : $\hat{X}$ and $\hat{Y}$, a maximal working range and width, $\bar{X}$ and $\bar{Y}$, a medium working range and width, $\hat{w}, \hat{h}$ minimum blob width and height, V, a vertical freedom threshold, λ, a threshold for the two-bases test, L, maximum horizontal distance and μ_1, μ_2, factors for calculating the final value for one or two shadows

Output : a real value between 0 and 1, representing the likelihood that this target is indeed of the type T and found in $\langle X, Y\rangle$.

Additional data : the list of blobs $b_1 \dots b_n$, the output of the preprocessing phase.

The function :

1. If the algorithm is deactivated, or $X > \hat{X}$, or $\hat{X} \geq X > \bar{X}$ and $\mid Y \mid > \hat{Y}$, or $\bar{X} \geq X > 0$ and $\mid Y \mid > \bar{Y}$, return 0.5

2. project the position $\langle X, Y\rangle$ on the image and obtain R^{global} and R^{rear} as explained in subsection 6.2.1

3. The two-bases check : consider the two sub-rectangles of R^{rear}, $\overleftarrow{R}$ and $\overrightarrow{R}$, defined as :
$left(\overleftarrow{R}) = left(R^{rear})$
$right(\overleftarrow{R}) = left(R^{rear}) + \frac{1}{4} width(R^{rear})$
$top(\overleftarrow{R}) = top(R^{rear}) + \frac{3}{4} height(R^{rear})$
$bottom(\overleftarrow{R}) = bottom(R^{rear})$
$left(\overrightarrow{R}) = left(R^{rear}) + \frac{3}{4} width(R^{rear})$
$right(\overrightarrow{R}) = right(R^{rear})$
$top(\overrightarrow{R}) = top(R^{rear}) + \frac{3}{4} height(R^{rear})$
$bottom(\overrightarrow{R}) = bottom(R^{rear})$
For a rectangle R we define as $val(R)$ the number of pixels in the thresholded image which are equal to 1 and fall inside the rectangle R.
If $\overleftarrow{R}$ is entirely inside the image and $val(\overleftarrow{R}) < surface(\overleftarrow{R}) * \lambda$ or $val(\overrightarrow{R}) < surface(\overrightarrow{R}) * \lambda$, then return 0.25

4. let $B \subseteq \{b_1 \dots b_n\}$ be the set of all blobs b for which the following conditions hold :

   4.1 $height(b) \geq \hat{h}$ or $width(b) \geq \hat{w}$

   4.2 if R^{global} is not entirely in the image, the blob b is touching one of the image borders (that is $left(b) = 0$ or $top(b) = 0$ etc.)

   4.3 $\mid bottom(b_i) - bottom(R^{global}) \mid \leq V$

5. define :
$\overleftarrow{\Delta} = min_{b \in B} min(\mid left(R^{rear}) - left(b) \mid, \mid left(R^{global}) - left(b) \mid)$
$\overrightarrow{\Delta} = min_{b \in B} min(\mid right(R^{rear}) - right(b) \mid, \mid right(R^{global}) - right(b) \mid)$

6. if $\overleftarrow{\Delta} > L$ then $\overleftarrow{\Delta} \leftarrow L$, same for $\overrightarrow{\Delta}$

7. If only the right (left) side of R^{global} is inside the image, then calculate the likelihood value according to the right shadow only :
result = $1 - \overrightarrow{\Delta} * \mu_1$ $(1 - \overleftarrow{\Delta} * \mu_1)$
otherwise, calculate according to both shadows :
result = $1 - (\overrightarrow{\Delta} + \overleftarrow{\Delta}) * \mu_2$

8. Clip the result in the range $[0.25, 1]$ and return it

TAB. 6.4 – The shadow detection likelihood function.

As in the other algorithms, the result is always clipped with the range $[0.25, 1]$. A probability below 0.25 cannot be evaluated by this algorithm, because it can never be sure that its analysis is perfect (step 11).

Initializing new targets

Targets are initialized when there is a blob which might represent a vehicle. Not entirely unlike the calculations we make in the likelihood function, we run the following tests for each blob to see if it corresponds to a real object :

1 We check, according to the camera parameters, if the width of the blob in the real world is approximately the width of a car, truck or motorcycle. To do that, we retro-project the blob to the scene six times - two times for each type (car, truck, motorcycle). The first time, assuming it is the shadow of the rear of the vehicle. The second, assuming it is the shadow of the entire vehicle. Both cases are identical if the vehicle is situated exactly in front of us. For each of these six retro-projections, we project back the six vehicles to the image, and see if we got the same vertical location (up to an error of 10 pixels). If the blob cannot represent that vehicle, the projection will be far from the blob, as seen in image 6.14.

2 We perform the two-bases check, to see if the blob is horizontal, as described above.

3 We only use blobs which are not touching the image borders. Blobs touching the image borders might correspond to vehicles entering the scene, but these types of vehicles will be initialized by motion estimation later. Shadow detection only initializes targets which are entirely on the scene. Using shadow detection to initialize entering vehicles produces too much noise because often shadows are created on the sides by sidewalks, fields, etc.

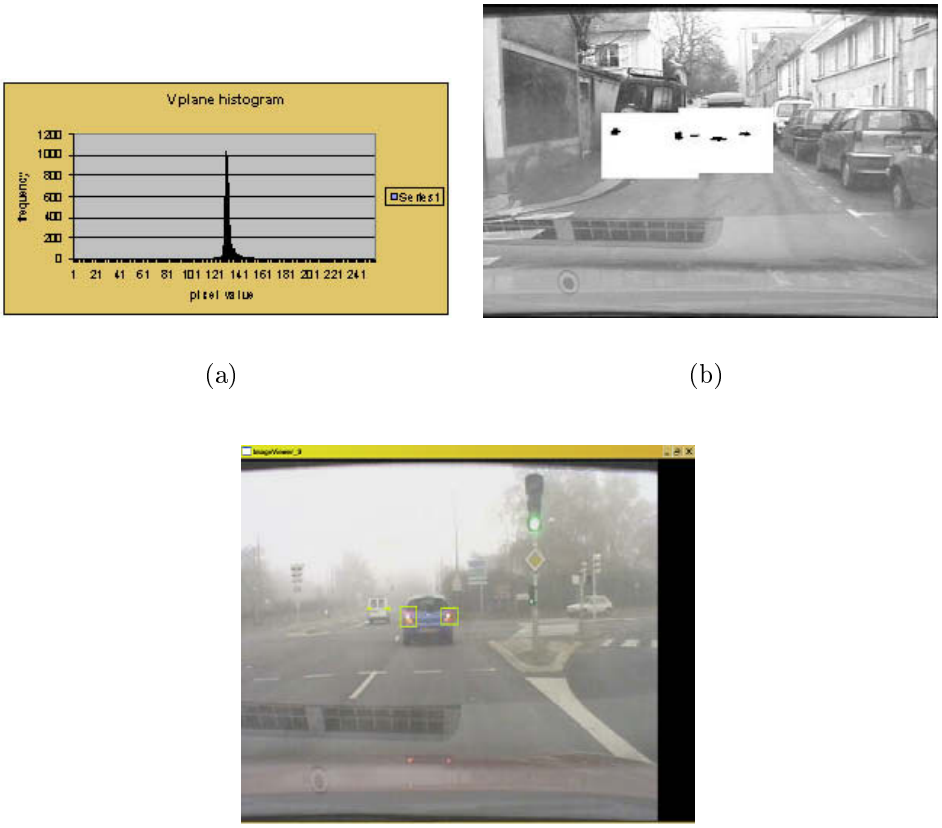
To initialize, we take each one of the retroprojections which passed test 1 above (it might be all the six retroprojections, some of them, or none), and spread 100 particles over its location, in a net structure. The result is up to 600 particles representing different kinds of vehicles in different positions, all according to the given blob. This is done for each blob which passed the 3 tests above.

6.2.3 Car's rear-lights detection

This algorithm uses the fact that cars and trucks usually have noticeable red rear lights, and a yellow license plate. The algorithm is using the V plane of the input image. High values in this plane represent red and yellow, and thus a simple threshold on this plane can reveal car's lights and plate. This algorithm is slightly less effective on motorcycles, where there is only one light, but it can also help in this case.

The algorithm uses information about targets which existed already in the previous step. For each existing target t, we take an image rectangle $R^t = \cup_{p \in t} proj(p)$, where $proj(p)$ is the global projected rectangle of the particle p.

The algorithm starts by computing the histogram of the V plane in the bounding rectangle R^t of each existing target t. For each rectangle, using the histogram, we choose a value which



(a) (b)

(c)

FIG. 6.9 – Lights detection algorithm. First, the histogram is calculated (a). Then, the region is thresholded using a value calculated from the histogram (b). The obtained blobs are used to detect the rear lights (c).

is superior of 98% of the pixels in that rectangle. This is demonstrated in figure 6.9. Typically, pixels having a larger value than this threshold will represent strong red or yellow intensities. Next we threshold each rectangle with its corresponding threshold value and obtain, ideally, a clear view of the lights and plate.

After the threshold, a blob analysis is performed on each rectangle and the resulting lists of blob (one list L^t for each existing target t) are the output of the preprocessing stage. We will see how these lists are used to calculate the likelihood.

The whole process is repeated one more time *for the whole image*. That is, the histogram, the threshold value and the thresholding are performed not on a specific region, but on the entire image. The resulting list of blobs L^{all} will be used to initialize new targets, as we will see later.

Table 6.5 gives the entire algorithm for the lights detection.

Input : an image I_k, coming from the V plane of the input YUV image at step k of the execution, and a list of image rectangular regions $r_1 \dots r_m$, each region r_i being associated with an existing target t_i.

Parameters : Histogram percentage γ, minimum threshold value T.

Output : For each target t_i a list of blobs $b_1^i \dots b_{n_i}^i$, plus a global list of blobs $b_1^0 \dots b_{n_0}^0$.

The algorithm :

1. Define r_0 are an image rectangular region consisting of the entire image.

2. For $i = 0 \dots m$ do :

2.1 Calculate the histogram $v_0^i, v_1^i \dots v_{255}^i$ of the pixel values in the region r_i (v_p^i is the number of times the pixel value p appears in r_i). From the histogram, calculate m, the minimal value such that $\sum_{p=0}^{m} v_p^i > \gamma * \sum_{p=0}^{255} v_p^i$

2.2 If $m < T$, then $m \leftarrow T$

2.3 Threshold I_k with the threshold value m

2.4 Erode the image with the following kernel e^{2x3} to remove noise :

$$e^{2x3} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

2.5 Analyze the resulting image and obtain the output list of connected bodies, or blobs, $b_1^i \dots b_{n_i}^i$. Each blob is represented by its bounding rectangle on the image.

TAB. 6.5 – The rear lights detection preprocessing phase.

The likelihood function

The input of the likelihood function, as in the previous algorithm, is a particle containing a hypothesis $\langle X, Y, T \rangle$ of an existing target t. The function is running on the list of blobs L^t discovered for this target in the preprocessing phase. Our goal is to find 1,2 or 3 blobs which will optimally match the vehicles lights and plate.

There are several cases, shown in figure 6.23 :

- In case of a motorcycle, we search for one blob, corresponding to the single rear light, which should be situated in the middle of the rear projected rectangle of the motorcycle.
- In case of a car or a truck which are not entirely on the scene, we are searching only for one light.
- In case of a fully visible car or truck, we are searching for two lights, and preferably also a plate. However, even if we don't find a plate, and even if one light is missing, we will give a moderate probability estimation, because sometimes the plate or one of the lights are not clear enough to appear on the thresholded image.
- In some cases the lights and even the plate are appearing as one long blob, in which case we should also give reasonable probability estimation.

To cover all these cases, we simply assign each blob its best "role" - that is, does it resemble a left light, a right light or a plate. To test if a blob corresponds to a left or right light, we :

- check if the top and bottom of the blob are within the limits shown in figure 6.20 ;
- check if the width of the blob is at least 1/10 of the width of the vehicle.

To test if a blob corresponds to a license plate, we :

- check if the top and bottom of the blob are within the limits shown in figure 6.20 ;
- check if the width of the blob is at least 1/8 of the width of the vehicle ;
- check if the blob is flat - its width must be at least 3 times larger than its height.

The entire likelihood function is given in table 6.6. After the matching operation we have :

1 b^l - a blob which is the most likely to be the left light - it complies with the relevant conditions above and minimizes the distance between the left of the blob and the left of the rear projected rectangle of the vehicle.

2 b^r - a blob which is the most likely to be the right light - it complies with the relevant conditions above and minimizes the distance between the right of the blob and the right of the rear projected rectangle of the vehicle.

3 b^l - a blob which is the most likely to be the plate - it complies with the relevant conditions above and minimizes the distance between the center of the blob and the center of the rear projected rectangle of the vehicle.

As seen in the table, we calculate a probability between 0 and 1 according to these distances. For vehicles which fall outside the image, we use only the distance which relates to the visible side.

The result is always clipped within the range $[0.25, 1]$.

Input : a particle containing a hypothesis $\langle X, Y, T\rangle$ about an existing target t_i, where $\langle X, Y\rangle$ is the position of this target and $T \in \{MOTORCYCLE, CAR, TRUCK\}$ is its type.

Parameters : a factor λ for the minimum result calculation, μ_1, μ_2 and μ_3, factors for calculating the final value for one or two lights and for a plate, a maximal value $\bar{M}$ for case of no plate, ρ - a penalty for two connected lights

Output : a real value between 0 and 1, representing the likelihood that this target is indeed of the type T and found in $\langle X, Y\rangle$.

Additional data : for each existing target $t_i, i = 1 \dots m$, a list of blobs $b_1^i \dots b_{n_i}^i$, the output of the preprocessing phase.

The function :

1. project the position $\langle X, Y\rangle$ on the image and obtain R^{global} and R^{rear} as explained in subsection 6.2.1

2. If R^{rear} is *entirely not* in the image, return 0.5

3. Determine the minimum possible result of this algorithm, $\underline{p}$:
$\underline{p} = min(0.5, 0.25 + X * \lambda)$

4. Find all blobs corresponding to lights : let $L \subseteq \{b_1^i \dots b_{n_i}^i\}$ be the set of all blobs b for which the following conditions hold :
   4.1 $width(b) > \frac{1}{10} width(R^{rear})$
   4.2 $top(R^{rear}) \leq top(b) \leq top(R^{rear}) + \frac{3}{4} height(R^{rear})$
   4.3 $top(R^{rear}) + \frac{1}{4} height(R^{rear}) \leq bottom(b) \leq bottom(R^{rear})$

5. Find all blobs corresponding to plates : let $P \subseteq \{b_1^i \dots b_{n_i}^i\}$ be the set of all blobs b for which the following conditions hold :
   5.1 $width(b) > \frac{1}{8} width(R^{rear})$
   5.2 $top(R^{rear}) + \frac{3}{8} height(R^{rear}) \leq bottom(b) \leq top(R^{rear}) + \frac{3}{4} height(R^{rear})$
   5.3 $width(b) > 3 * height(b)$

6. define :
   6.1 $\overleftarrow{\Delta} = min_{b \in L} \mid left(R^{rear}) - left(b) \mid$
   6.2 $\overrightarrow{\Delta} = min_{b \in L} \mid right(R^{rear}) - right(b) \mid$
   6.3 $\bar{\Delta} = min_{b \in P} \mid left(R^{rear}) - left(b) \mid$

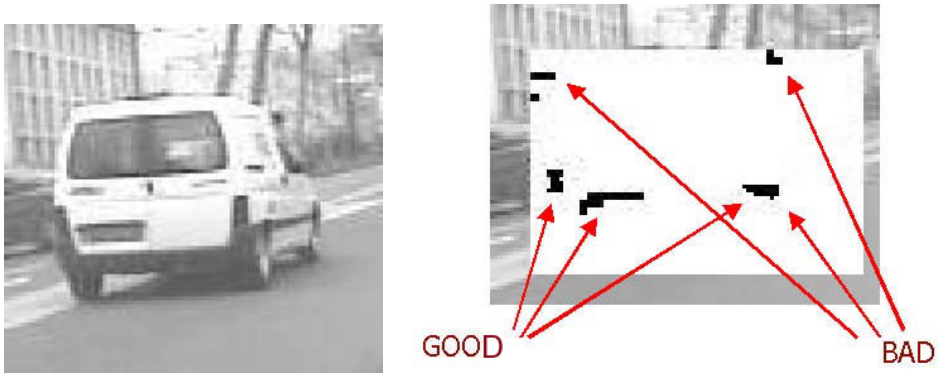
7. if the blob of the minimum in 6.3 is the same one as the blob of the minimum of 6.1 or 6.2, then $\bar{\Delta} = \bar{M}$

8. If only the right side of R^{rear} is inside the image, then calculate the likelihood value according to the right light and possible plate only :
result $= 1 - \overrightarrow{\Delta} * \mu_1 + \bar{\Delta} * \mu_3$
If only the left side of R^{rear} is inside the image, then calculate the likelihood value according to the left light and possible plate only :
result $= 1 - \overleftarrow{\Delta} * \mu_1 + \bar{\Delta} * \mu_3$
otherwise, calculate according to both shadows and possible plate :
result $= 1 - (\overrightarrow{\Delta} + \overleftarrow{\Delta}) * \mu_2 + \bar{\Delta} * \mu_3$

9. if the blob of the minimum in 6.1 is the same one as the blob of the minimum of 6.2, then :
$result \leftarrow result - \rho$
except if $T = MOTORCYCLE$

10. Clip the result in the range $[0.25, 1]$ and return it

TAB. 6.6 – The rear lights detection likelihood function.



(a) (b)

FIG. 6.10 – The selection of the blobs for lights detection.

6.2.4 Symmetry detection

Symmetry detection is based on calculating the SAD (Sum of Absolute Differences) between some area of the image and its adjacent area in the mirror image, as seen in Figure 6.11. If the two areas are two parts of a symmetry, the SAD will be zero.

For each existing target, we repeat the calculation of the SAD on a set of adjacent pairs of regions and obtain a symmetry function as shown in Figure 6.11. The symmetry function of a car typically has the "MacDonald" M-shape as shown in the image. This is due to the fact that the car is symmetric in the middle, but its sides are anti-symmetric. Note that the low symmetry on the sides is as important as the symmetry at the middle. To understand this, consider the sky, which has good symmetry in all the places.

The results are used only to provide likelihood value for particles of existing targets. This algorithm does not initialize new targets.

The processing phase is listed in table 6.7. The output of the preprocessing phase is a symmetry function for each existing target.

The likelihood function

Basically, we would like to give a high likelihood value if the symmetry function resembles the "McDonald" shape. That is :

1 is having two maximum points located horizontality near the left and right sides of the vehicle (i.e. its rear projected rectangle).

2 is having a minimum located horizontally near the center of the vehicle.

3 these points should be regional, not local, extremum points. That is, they must be larger/smaller than the majority of their environment.

4 the derivative of the function must take the form up-down-up-down-up.

Input : an image I_k, coming from the Y plane of the input YUV image at step k of the execution, and a list of image rectangular regions $r_1 \dots r_m$, each region r_i being associated with an existing target t_i, and a list of corresponding width values $w_1 \dots w_m$ of typical width of the target

Parameters : .

Output : For each target t_i a symmetry function (sequence of integers) $f_0^i \dots f_{width(r_i)-2*w_i}^i$

The algorithm :

1. Calculate the mirror image I'_k from I_k.
We will use the notion $val(x, y)$ to represent the pixel value in point $\langle x, y \rangle$ in the image I_k, and $val'(x, y)$ for the pixel value in the *mirrored point of* $\langle x, y \rangle$ in the mirrored image I'_k

2. For $i = 1 \dots m$ do :

2.1 For each $j = 0 \dots width(r_i) - 2 * w_i$, define rectangles R_j^{left} and R_j^{right} as follows :

$left(R_j^{left}) = left(r_i) + j$
$right(R_j^{left}) = left(r_i) + w_i - 1 + j$
$left(R_j^{right}) = left(r_i) + w_i + j$
$right(R_j^{right}) = left(r_i) + 2 * w_i - 1 + j$
$top(R_j^{left}) = top(R^{right}) = top(r_i)$
$bottom(R_j^{left}) = bottom(R^{right}) = bottom(r_i)$

2.2 If R_j^{left} or R_j^{right} are not entirely inside the image, deactivate the algorithm *for this target* and continue (back to step 2)

2.3 For each $j = 0 \dots width(r_i) - 2 * w_i$, define the output :

$f_j^i = SAD(R^{left}, R^{right})$

Where for two rectangles R_1 and R_2 having the same width and height, we have :

$SAD(R_1, R_2) = \sum_{y_1=top(R_1)\dots bottom(R_1), y_2=top(R_2)\dots bottom(R_2)}$
$\sum_{x_1=left(R_1)\dots right(R_1), y_2=right(R_2)\dots left(R_2)} \mid val(x_1, y_1) - val'(x_2, y_2) \mid$

TAB. 6.7 – The symmetry algorithm preprocessing phase.

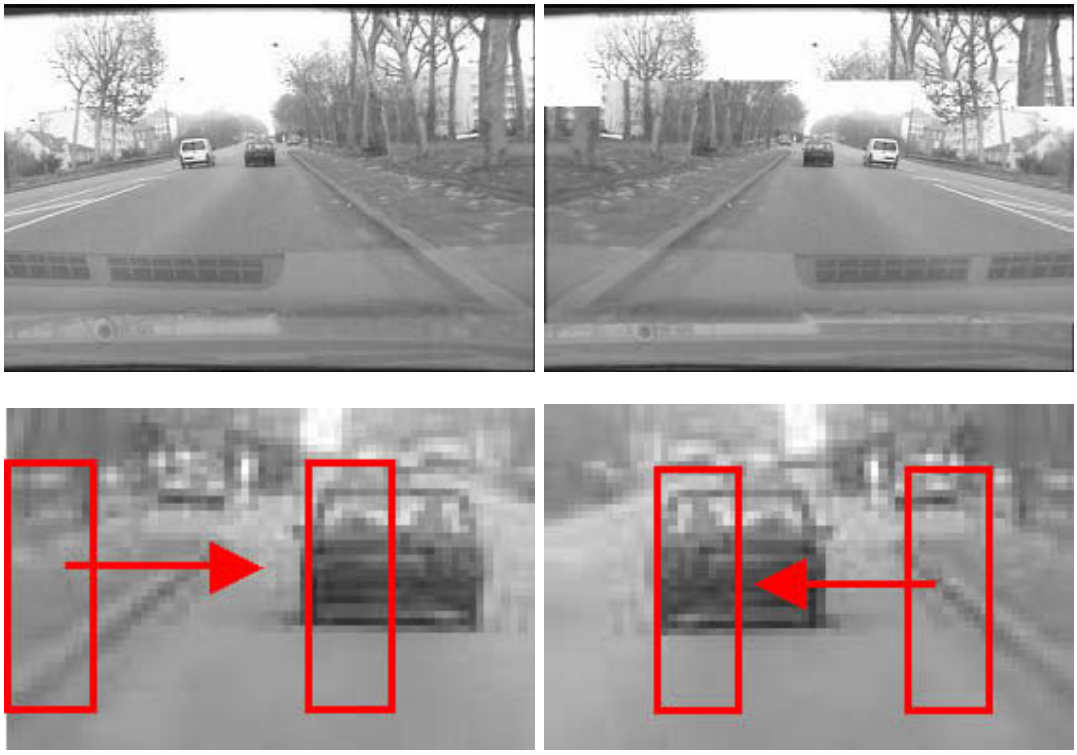


FIG. 6.11 – The Symmetry detection. From top to bottom : the original image, the mirrored image, left and right rectangles.

The likelihood function, given in tables 6.8 and 6.9, is searching for three such extremum points. It searches for the points in the environment of the location where it expects them to be - that is, on the left, center or right of the vehicle.

For each of the 3 points, it tests the 3 conditions above (conditions 1,3,4 for the left and right, and conditions 2,3,4 for the center), and gives each point a compatibility mark. The final likelihood function considers the lowest compatibility mark of the three points. This assures that only real "McDonald" shapes will get high likelihood.

Figures 6.24 and 6.25 show several symmetry functions and their likelihood value.

6.2.5 Vertical edges detection

Vertical edges detection is based on looking for typical patterns of the vertical summing of the Sobel image. The Sobel image, like in the example given in figure 6.13, is created from the original image by linear filtering with the following kernel :

$$e^3 = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

One can clearly see that cars and trucks have two strong vertical edges on their sides. If we vertically sum these values and visually present the obtained function, we will see two peaks at the sides of the vehicle, as shown in figure 6.13. The two peaks become three when the car is diagonally positioned.

Our algorithm is based on detecting these peaks. Like in the symmetry algorithm, this algorithm does not initialize new targets, and provides the likelihood value based on the vertical edges function. The algorithm is given in table 6.10. For each existing target, we calculate the vertical edges function, which is the output of the preprocessing phase.

The likelihood function

Given a particle, we project it on the image and create the "ideal" function, according to the type and location of the vehicle. To obtain the likelihood value, we convolute the function with an "ideal" function. The convolution yields a number between 0 and 1, according to the similarity of the vertical edges function to the ideal one.

The likelihood function is given in table 6.11.

6.2.6 Detection with AdaBoost

In chapter 3 we give results of cars detection by AdaBoost. These results are used here. This algorithm contains no preprocessing stage, and it does not initialize targets. It simply provides likelihood value for existing targets, as described below.

The likelihood function

Given a particle with an hypothesis $\langle X, Y, T \rangle$, we first check its type T. According to the type, we employ the correct AdaBoost classifier previously trained for this type of objects.

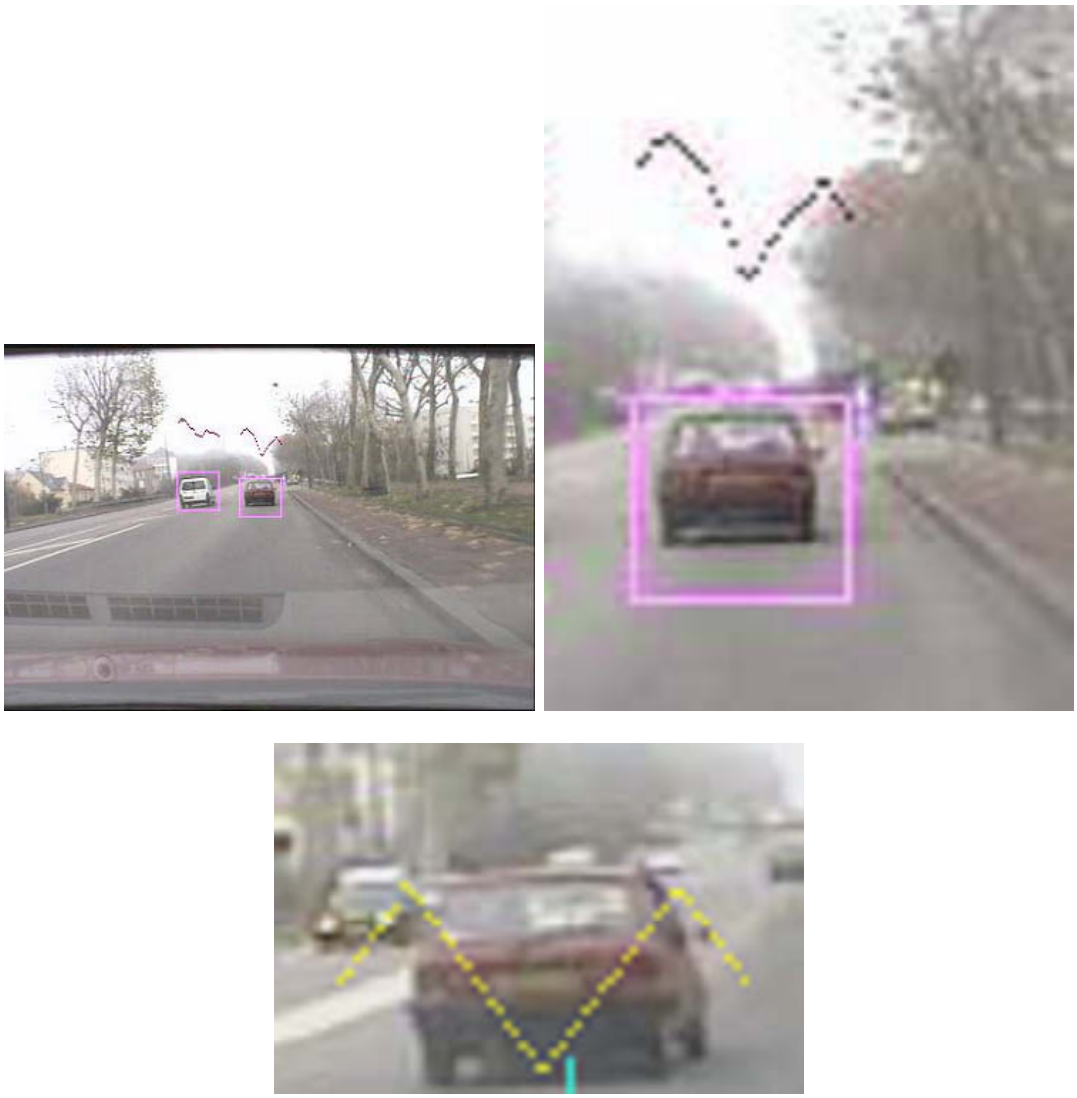


FIG. 6.12 – The actual symmetry function, general view (top) and closer look (middle), and the synthetic function (bottom)

Input : a particle containing a hypothesis $\langle X, Y, T\rangle$ about an existing target t_i, where $\langle X, Y\rangle$ is the position of this target and $T \in \{MOTORCYCLE, CAR, TRUCK\}$ is its type.

Parameters : a factor λ for the left, center and right point calculation

Output : a real value between 0 and 1, representing the likelihood that this target is indeed of the type T and found in $\langle X, Y\rangle$.

Additional data : For each existing target t_i a symmetry function (sequence of integers) $f_0^i \dots f_{width(r_i)-2*w_i}^i$, the output of the preprocessing phase.

The function :

1. If the algorithm is deactivated for t_i, return 0.5

2. project the position $\langle X, Y\rangle$ on the image and obtain R^{global} and R^{rear} as explained in subsection 6.2.1

3. Let $P^{left} = left(R^{rear}), P^{center} = \frac{left(R^{rear})+right(R^{rear})}{2}, P^{right} = right(R^{rear})$ (the expected left,center and right of the expected "McDonald" shape), and let $W = \frac{1}{2}width(R^{rear})$ (the expected width of each "leg" of the expected "McDonald" shape)

4. Search for a the left peak of the expected shape. For each $i = P^{left} - \frac{W}{4} \dots P^{left} + \frac{W}{4}$, calculate :

4.1 $\alpha_i^{left} = CalculateExtremumMark_{TOP,\frac{W}{4}}(i)$ the mark of this point in terms of extremum (should be higher than most of its neighbors)

4.2 $\beta_i^{left} = CalculateDerivativeMark_{TOP,\frac{W}{4}}(i)$ the mark of this point in terms of derivative (should be mostly increasing before the point and decreasing after it)
the definition of the auxiliary functions is found in table 6.9.

5. The marks α_i^{left} and β_i^{left} express the compatibility of the point i to be the left peak of the "McDonald" shape. The total compatibility mark considers the minimum of the 2 marks and also the distance of the point from the expected location P^{left} :
$\delta_i^{left} = min(\alpha_i^{left}, \beta_i^{left}) - \mid i - P^{left} \mid *\lambda$

6. We choose the best candidate $\hat{P}^{left}$ as the position i maximizing the mark δ_i^{left}. Define the best obtained mark $\delta_{\hat{P}^{left}}^{left}$ as $\hat{\delta}^{left}$

7. Repeat steps 4-6 for the center point (with a "BOTTOM" parameter for the 3 auxiliary functions) and right point (with a "TOP" parameter) of the "McDonald" shape and obtain $\hat{P}^{center}$,$\hat{\delta}^{center}$ and $\hat{P}^{right}$,$\hat{\delta}^{right}$.

8. Let the final likelihood value be :
$result = 1 - \frac{1}{100}min(\hat{\delta}^{left}, \hat{\delta}^{center}, \hat{\delta}^{right})$

TAB. 6.8 – The symmetry likelihood function

Auxiliary functions, calculating the compatibility of a point to be a top/bottom point in a "McDonald" shape, according to different criteria. All 3 functions have an input location p and parameter $T \in \{TOP, BOTTOM\}$ (wether to search for top or bottom point) and search width $\hat{W}$.

$CalculateExtremumMark$ is testing the compatibility in terms of extremum point. If $T = TOP$ we return :
$\frac{|\{j | p-\hat{W} \leq j \leq p+\hat{W}, f_p^i > f_j^i\}|}{2*\hat{W}}$
if $T = BOTTOM$ return
$\frac{|\{j | p-\hat{W} \leq j \leq p+\hat{W}, f_p^i < f_j^i\}|}{2*\hat{W}}$

$CalculateDerivativeMark$ is testing the compatibility in terms of derivative environment. If $T = TOP$ we return :
$\frac{|\{j | p-\hat{W} \leq j \leq p+\hat{W}, sign(f_j^i - f_{j-1}^i) \neq sign(j-p)\}|}{2*\hat{W}}$
if $T = BOTTOM$ return
$\frac{|\{j | p-\hat{W} \leq j \leq p+\hat{W}, sign(f_j^i - f_{j-1}^i) = sign(j-p)\}|}{2*\hat{W}}$

TAB. 6.9 – The symmetry auxiliary functions

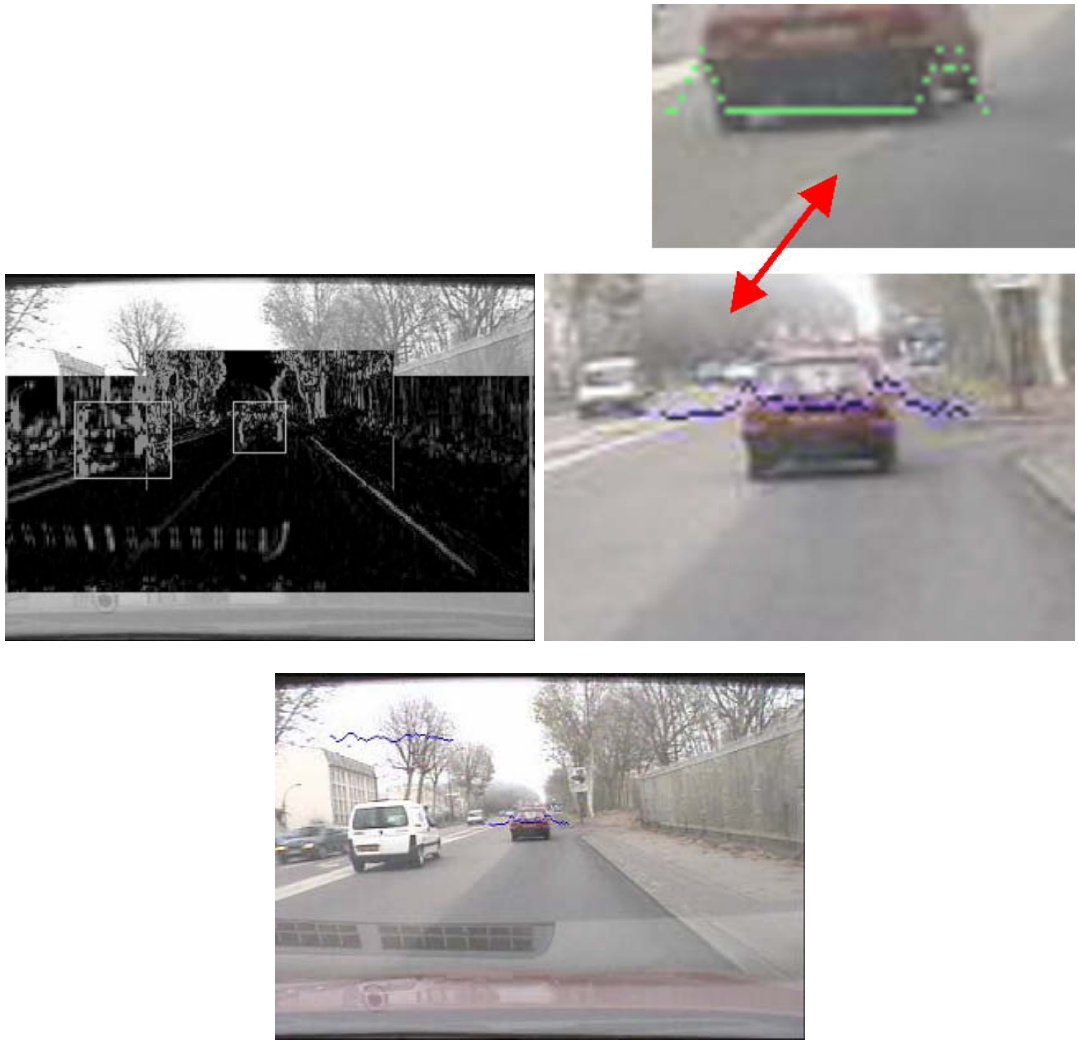


FIG. 6.13 – The edges detection process. From top to bottom : the Sobel image, the resulting vertical edges function, and the synthetic function.

Input : an image I_k, coming from the Y plane of the input YUV image at step k of the execution, and a list of image rectangular regions $r_1 \dots r_m$, each region r_i being associated with an existing target t_i.

Parameters : .

Output : For each target t_i a vertical edges function (sequence of integers) $f_0^i \dots f_{width(r_i)-1}^i$

The algorithm :

1. For $i = 1 \dots m$ do :

1.1 If r_i are not entirely inside the image, deactivate the algorithm *for this target* and continue (back to step 1)

1.2 Apply the linear filtering on the region r_i with the following element :

$$e^{2x3} = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

1.2 For each $j = 0 \dots width(r_i) - 1$, define the output :

$$f_j^i = \sum_{p=top(r_i)}^{bottom(r_i)} val_{I_k}(j, p)$$

where $val_{I_k}(x, y)$ is the pixel value in the image I_k at point $\langle x, y \rangle$

TAB. 6.10 – The vertical edges algorithm preprocessing phase.

Input : a particle containing a hypothesis $\langle X, Y, T\rangle$ about an existing target t_i, where $\langle X, Y\rangle$ is the position of this target and $T \in \{MOTORCYCLE, CAR, TRUCK\}$ is its type.

Parameters : Ξ, the margins of the synthetic function, $\bar{K}$, a factor for final calculation of the likelihood value

Output : a real value between 0 and 1, representing the likelihood that this target is indeed of the type T and found in $\langle X, Y\rangle$.

Additional data : For each existing target t_i a vertical edges function (sequence of integers) $f_0^i \dots f_{width(r_i)-1}^i$, the output of the preprocessing phase.

The function :

1. If the algorithm is deactivated for t_i, return 0.5

2. project the position $\langle X, Y\rangle$ on the image and obtain R^{global} and R^{rear} as explained in subsection 6.2.1

3. Define the 3 visible edges of the vehicle $p_1, p_{2,3}$. If $left(R^{global}) < left(R^{rear})$, then :
$p_1 = left(R^{global})$
$p_2 = left(R^{rear})$
$p_3 = right(R^{rear})$
otherwise :
$p_1 = left(R^{rear})$
$p_2 = right(R^{rear})$
$p_3 = right(R^{global})$

4. Create the synthetic function $v_1 \dots v_m$, where $m = p_3 - p_1 + 2 * \Xi$:
$d_i = min_{i=1,2,3} \mid i - p_i \mid$
$e_i = \hat{M} - d_i * \tilde{M}$
$v_i = \begin{cases} e_i & \text{if } e_i > 0 \\ k & \text{otherwise} \end{cases}$
where $k = -\frac{\sum_{i=0}^{m} max(e_i, 0)}{|\{i | e_i < 0\}|}$
Note that the k value is calculated such that the $\sum_{i=0}^{m} v_i = 0$

5. find the portion of the vertical edges function that we are going to use :
$u_{left} = p_1 - \Xi - left(r_i)$
$u_{left} = p_3 + \Xi - left(r_i)$

6. Normalize this portion of the vertical edges function :
$\forall j = u_{left} \dots u_{right}, f_j^i \leftarrow \frac{f_j^i}{\sum_{j=u_{left}}^{u_{right}} f_j^i}$

7. Calculate the convolution :
$conv = \prod_{j=u_{left}}^{u_{right}} f_j^i * v_j$

8. Calculate the final result as :
$result = \frac{conv}{K}$

9. Result is clipped with the range $[0.25, 1]$

TAB. 6.11 – The vertical edges likelihood function.

At the time of writing this document, no training results are available for motorcycle, so if $T = MOTORCYCLE$, we return 0.5. Otherwise, for cars and trucks, we use the vehicle training described in chapter 3.

We project to the image a 8-point box whose center bottom rear point is $\langle X, Y \rangle$. Then, we take the rear projected rectangle and warp it to dimensions 36x24 - the dimensions of the training of cars. Then we apply our strong classifier and *return the boosting value as the likelihood value*. This technique allows smooth evaluation instead of yes/no style answer.

If the result is less than 0.1, it is fixed on 0.1, since 10% is the possible error of the algorithm, according to our estimation.

6.2.7 Weakness of algorithms

Algorithms do not always provide us with perfect results. In some conditions, it might be better to deactivate some of them. For example, over a certain distance, some algorithms do not work anymore and their output can be considered as noise. In this section, we will describe the limitations of each algorithm.

Shadow detection

The shadow detection algorithm depends primarily on adequate lighting conditions. Without such conditions, we might observe effect which resemble shadows, but in fact these will not be shadows. In order to avoid these conditions, as mentioned in table 6.3, the algorithm deactivates itself when the variance of the "road ROI" drops below a certain threshold.

Symmetry and vertical edges detection

The main problem with these algorithms arises on the sides of the image. In these areas the objects are less symmetric and have less vertical edges, due to the different positions of the car. As a result, the symmetry and vertical edges measurements are not reliably expressing the desired likelihood. To avoid this, we deactivate the symmetry and the vertical edges algorithms if the target is found in the sides, as defined in tables 6.10 and 6.7.

Lights detection

The weakness of the light detection algorithm is mainly in the fact that lights are not red enough to be distinguished from their neighborhood. It was seen in experiments, that above a certain distance, the lights signal contains mainly noise and cannot be used. Therefore, the algorithm is deactivated above certain distance, as described in table 6.5.

6.3 Additional algorithms

6.3.1 Motion segmentation as a tool for fast detection of cut-ins

This algorithm uses the motion detector of [Witt 01], which is highly efficient. This motion segmentation algorithm is fully described in chapter 4. The system computes motion models

for each image block, then groups blocks with similar motion, to form objects. These objects pass through a temporal filter that keeps only the motions which are consistent for several consecutive images.

The algorithm is used to initialize targets and to provide likelihood values. The calculation is done simply by correlation with the detected moving objects. As will be shown, the likelihood value of this algorithm is quite coarse ; its important value is in initializing targets.

The motion estimator is acting on the current image I_k and on the previous one I_{k-1} (because of this, the entire application starts working in the second image). The estimator is providing us with a list of blocks $b_1 \dots b_m$ which represent moving objects in the scene. Each object contains its bounding rectangle and a motion vector in screen coordinates.

Likelihood function

Given a particle, we project it on the image and obtain the global projected rectangle R^{global}. The likelihood value is simply the amount of intersection between this rectangle and some moving object. However, we take only object which are moving to the image center, so the background (trees, etc.) is not taken into account.

The likelihood function is given in table 6.12. Note that we do not give less than 0.5 as likelihood. This is because objects are allowed not to move.

6.3.2 Integration with LIDAR

The lidar is the only non-vision source in our system. It is a device which gives a more precise information, but has a limited angle of view. For this subsection, it is enough to say that the lidar's input to our system is a list of detected points $\langle \dot{x}_1, \dot{y}_1 \rangle \dots \langle \dot{x}_m, \dot{y}_m \rangle$ in world coordinates. We use these points both for initializing new targets and for estimating likelihood of hypotheses about existing targets.

Likelihood of an hypothesis $\langle X, Y, T \rangle$ is using the following formula (result is clipped in the range $[0.25, 1]$) :

$result = max_{i=1\dots m}(1 - \mid Y - \dot{y} \mid * \epsilon^y - \mid X - \dot{x} \mid * \epsilon^x$

The factors ϵ^y and ϵ^x are the accuracy factors of the lidar. In our implementation we chose $\epsilon^y = 1$ and $\epsilon^x = 0.25$.

Initialization of targets by lidar is simple. All we do is take the detection points which are not in proximity to an existing target, and create a new target in that place. For each such point $\langle \dot{x}, \dot{y} \rangle$ we create a target with the following PDF :

$p(\langle X, Y, T \rangle) = \frac{1}{3} e^{-\epsilon^x * (X - \dot{x})^2 + \epsilon^y * (Y - \dot{y})^2}$

uniformly between motorcycle, car and truck.

We sample this PDF by 300 particles, so for each vehicle type there are 100 particles concentrated around the point $\langle \dot{x}, \dot{y} \rangle$.

Input : a particle containing a hypothesis $\langle X, Y, T\rangle$ about an existing target, where $\langle X, Y\rangle$ is the position of this target and $T \in \{MOTORCYCLE, CAR, TRUCK\}$ is its type.

Parameters : A margin value M, an opening angle $0 \leq \alpha < 180°$

Output : a real value between 0 and 1, representing the likelihood that this target is indeed of the type T and found in $\langle X, Y\rangle$.

Additional data to use : a list of blocks $b_1 \ldots b_m$ which represent moving objects in the scene. Each object b contains its bounding rectangle $rect(b)$ and a motion vector $motion(b) = \langle motion_X(b), motion_Y(b)\rangle$ in screen coordinates.

The function :

1. project the position $\langle X, Y\rangle$ on the image and obtain R^{global} and R^{rear} as explained in subsection 6.2.1

2. let $B \subseteq \{b_1 \ldots b_m\}$ be the set of all moving objects b for which the following conditions hold :

   2.1 if $right(rect(b)) > C - M$, then $motion_X(b) > 0$

   2.2 if $left(rect(b)) < C + M$, then $motion_X(b) < 0$

   2.3 $\mid \frac{motion_X(b)}{motion_Y(b)} \mid > \tan\frac{1}{2}\alpha$

Where $C = \frac{1}{2} width(I_k)$ is the horizontal center of the image

3. Calculate the result by :
$result = max_{b \in B} \frac{surface(R^{global} \cap b)^2}{surface(R^{global}) * surface(b)}$

4. If $result < 0.5$, then $result \leftarrow 0.5$

TAB. 6.12 – The motion detection likelihood function.

TAB. 6.13 – The dimensions of vehicle types.

Type	Width	Length	Height
Car	1.70	3.70	1.43
Truck	3.20	6.50	2.80
Motorcycle	0.50	1.30	0.90

FIG. 6.14 – Examples for cases where the shadow rectangle do not provide a homogenous histogram.

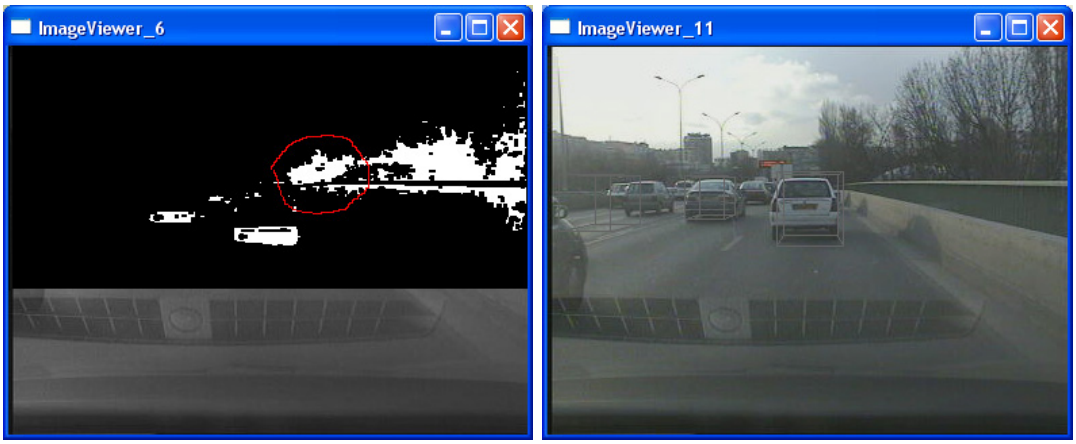


FIG. 6.15 – A shadow which cannot represent a vehicle, according to camera parameters.

FIG. 6.16 – Two forms of shadows : the global projection and the rear projection.

FIG. 6.17 – Shadows which might resemble a vehicle without the two-bases test.

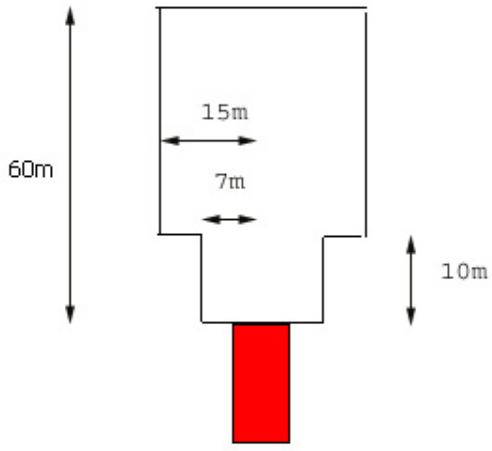


FIG. 6.18 – The limits of the shadow detection algorithm.

FIG. 6.19 – The two-bases test.

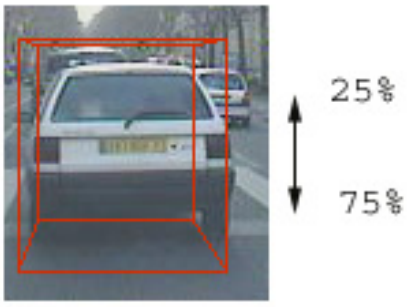


FIG. 6.20 – The limits of the location of the lights.

FIG. 6.21 – Use of only one side of a blob.

FIG. 6.22 – The shadow shape of a cut-in vehicle.

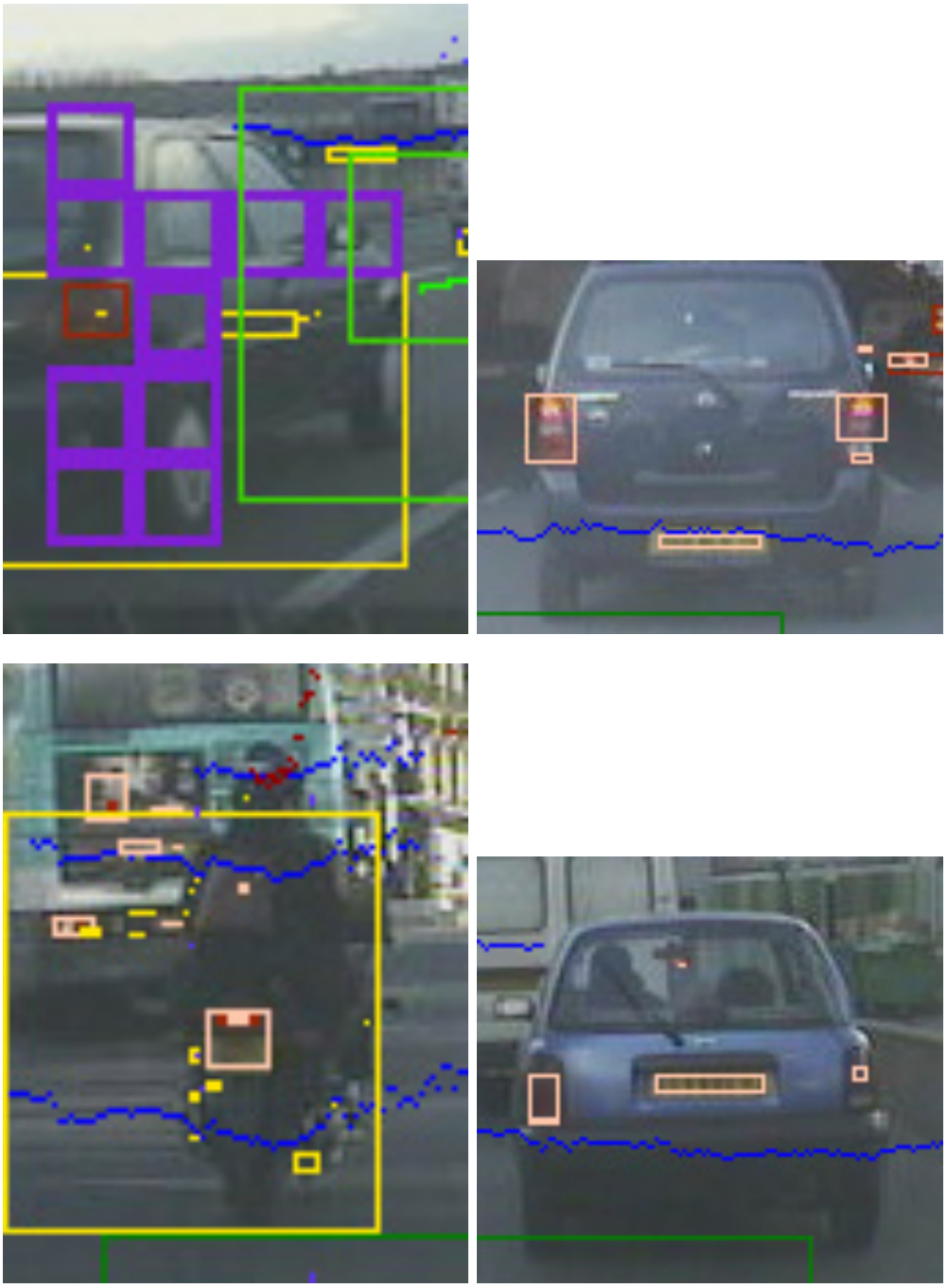


FIG. 6.23 – Different cases of lights.

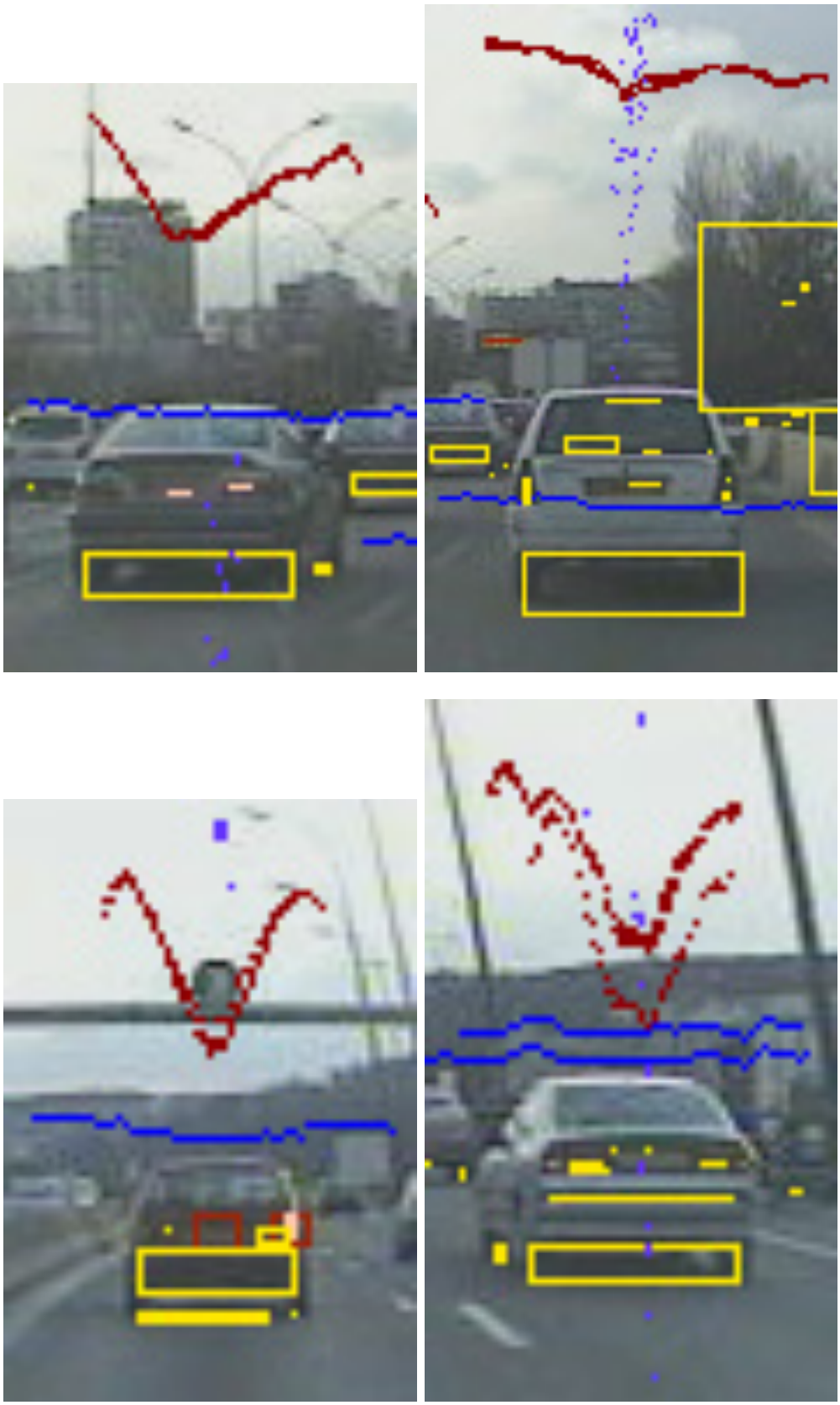


FIG. 6.24 – Different likelihood functions.

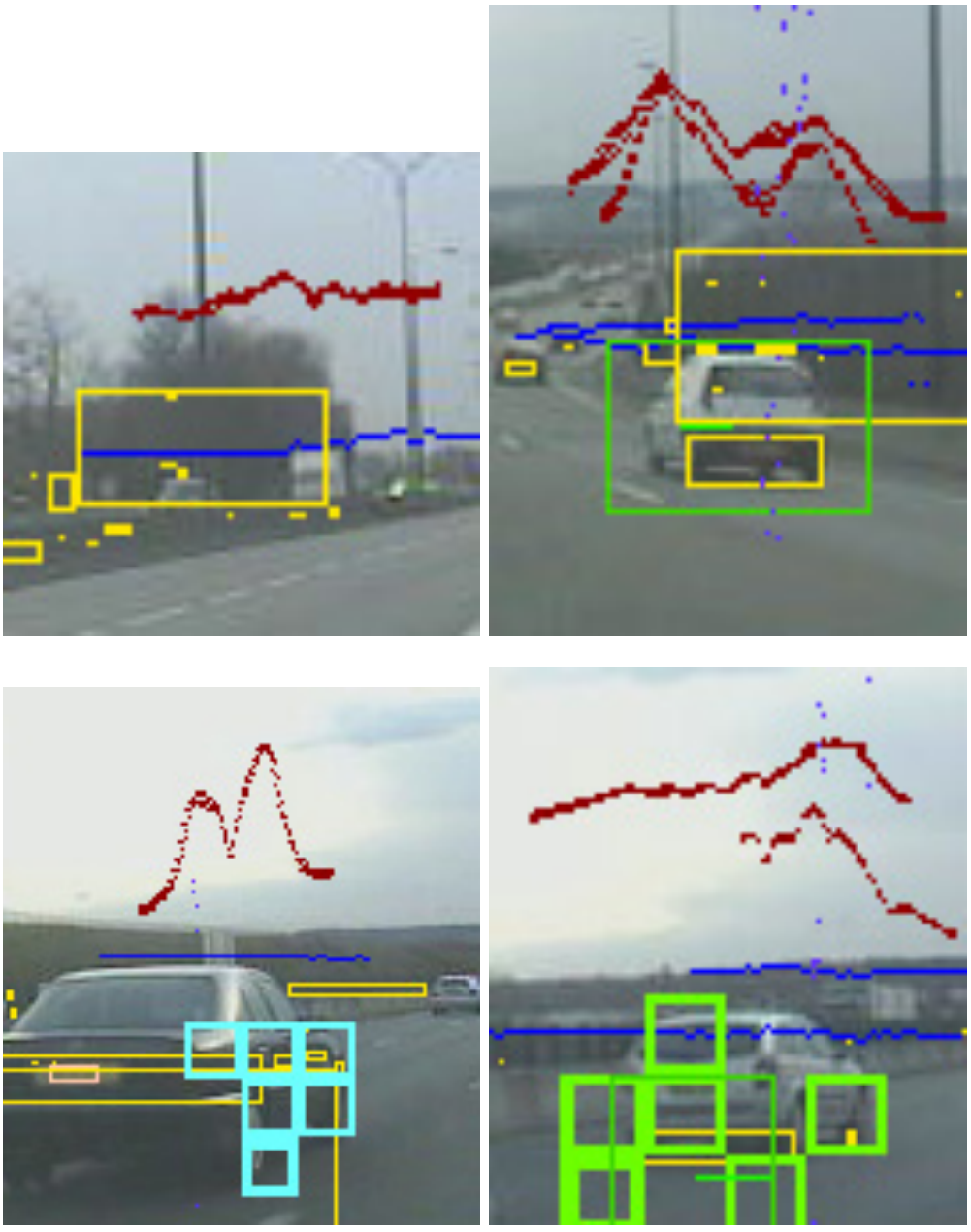


FIG. 6.25 – Different likelihood functions.

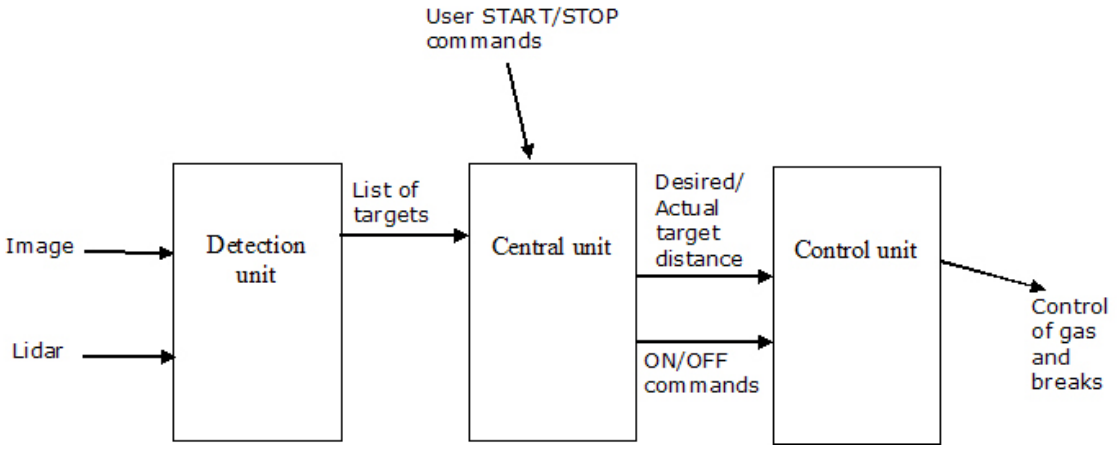


FIG. 6.26 – Architecture of the application.

6.4 The Stop&Go system

6.4.1 System architecture

As seen in figure 6.26, the entire application is made of three parts :

- the detection unit, described in section 6.1. Its output is a list of targets ;
- the *central unit*, which is using the results of the detector to lock on a frontal target and decide when to activate or deactivate the system ;
- the *control unit*, which is actually controlling the gas and breaks of the host vehicle, if needed.

The central unit is the one which "locks" on a target. As seen in the figure, the input of the central unit is :

- a list of targets detected by the detector ;
- a "START"/"STOP" request from the user.

The output of the central unit is :

- desired distance to target (which is actually the target's distance when a "START" signal was accepted) ;
- actual distance to target (according to the detector) ;
- "ON"/"OFF" command to the automatic control.

The central unit is fully described in subsections 6.4.2 and 6.4.3.

The control unit is the one which receives the actual and desired distance to target, and translates it into gas/break controls. The unit is activated and deactivated by the "ON" and "OFF" commands given to it by the central unit.

At the beginning, the car's gas/breaks are not controlled by the application and will not be so until the control unit is activated by receiving an "ON" command. An "OFF" command cuts the controlling of the car by the application.

The control unit input is :

- "ON"/"OFF" commands to activate and deactivate the control ;
- desired distance to target ;

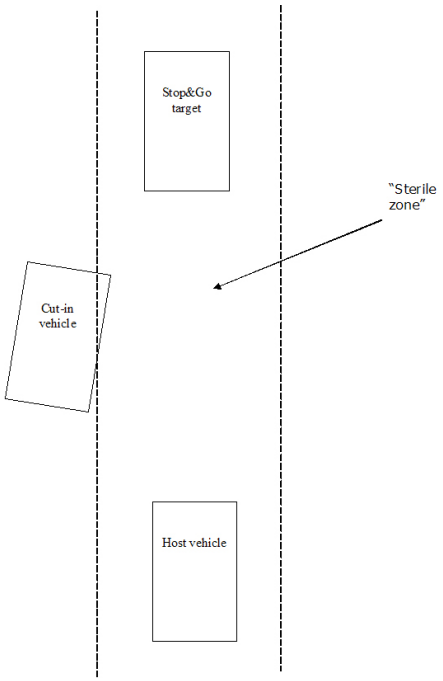


FIG. 6.27 – A cut-in situation.

– actual distance to target.

The output of the control unit is control commands it sends to the gas and breaks.

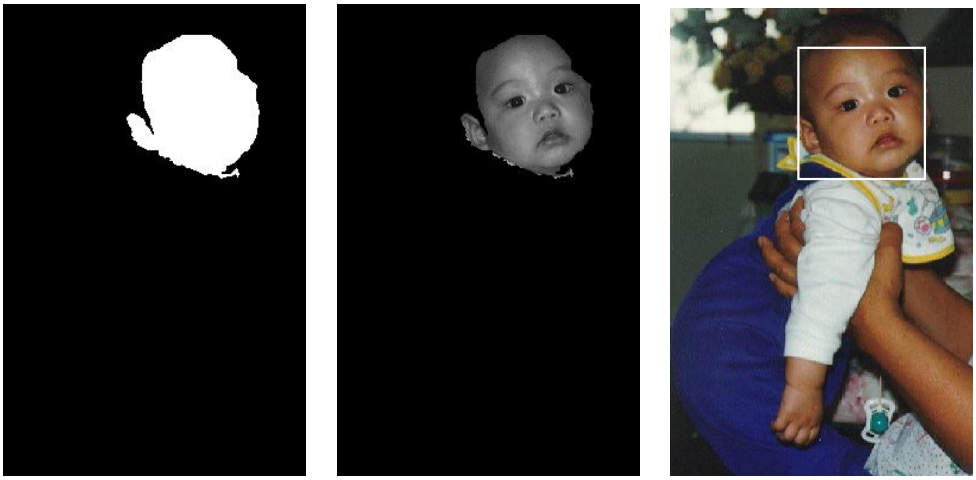
6.4.2 Identifying the Stop&Go target

The system is being activated by the driver, with a "START" command. Once this is done, the system runs the detection unit. After several images (needed for the detector to gain targets' confidence) we have some detections of targets on the scene. The first thing to do now is to locate the Stop&Go target. This target should be located in our lane ($\|y\| < 2$) and not too far $x < \hat{X}$. The value $\hat{X}$ should not be too high. Typically, such system is designed to be activated in short ranges, and the value we chose is $\hat{X} = 15m$. This doesn't mean that a target, after it is chosen, cannot get to be far than 15 meters. But the distance of the initial "lock" on the target should be close-ranged because we want to avoid the influence of curves.

If indeed, when activated, the application succeeds to lock on a target, it issues a "ON" command and starts sending desired and current target distance to the control unit. Otherwise (if a target could not be found) it does nothing, and the car retains driver's control. In such a case we say the system did not succeed to initialize. Driver could try again later to initialize the system, if he thinks a target is available.

6.4.3 Automatic disconnection of the system

If a target was found and an "ON" command was sent to the car, then the system is longitudinally controlling the car. At this time, if the driver is giving a "STOP" command



(a) (b) (c)

FIG. 6.28 – Template matching : (a) the mask (b) the part of the image within the mask, and (c) the entire image
(source : http ://ise.stanford.edu/class/ee368a_proj00/project16/main.html).

(by pressing a key), the central unit will immediately send a corresponding "OFF" command to the control unit. However, the central unit might do this even when driver did not stop the system. We will now discuss the cases in which it happens.

The system should stop itself as soon as it suspects that something has gotten between the host vehicle and the target vehicle. In such a case, the system must, of course, be stopped because this "something" might be an object which risks us. There are two ways in which the system can do this :

- when detecting a cut-in vehicle target which is entering the space between us and the Stop&Go target ;
- when noticing that our view of the Stop&Go target is not clear, may be because another object is hiding it. This is done using template matching.

These two ways will be now discussed.

Cut-in targets

Consider a situation like the one shown in figure 6.27. We are following a target, and the system detects another target which is entering into "the sterile zone" between us and the target vehicle. This "sterile zone" is basically our lane, in the part between us and the target vehicle. A vehicle is considered to have entered this space if *any part of its body is inside the space*. That is, as seen in the figure, even if the left of the entering car is in the sterile zone (and not its center) it's considered to have broken into this zone. In such a case, the central unit is sending an "OFF" command to the control unit.

Of course, one could say that in such a case we could have tried to acquire a new Stop&Go target - the vehicle that was entering this space. However, this is hard to perform ; we estimate

that it will not improve the application, but will make it less useful and more dangerous.

Template matching

The method described above is useful - we disconnect the system once another car has penetrated the space between us and the target. But what if it's not a car that penetrated ? what if it's an animal or an object falling from a truck ? And what if there is simply a failure of the detection system to continuing tracking the stop&Go target ? For all these cases we must have a mechanism that will alert us at the moment that we stop to obtain a clear image of our target.

Template matching is a classical method for object tracking. The main idea is to tailor a mask which will contain all the pixels which remain similar over consecutive images. In the case of figure 6.28, for example, the mask contains the details of the face of the baby. Assuming that after image I_k we already have such a mask (we will later see how such a mask is prepared ; meanwhile we donate the set of locations in the mask by M), then in order to locate the face of the baby in image I_{k+1} we can search several offsets $\langle dx_1, dy_1\rangle \dots \langle dx_n, dy_n\rangle$ and calculate for each of them the sum of absolute differences with the previous image :

$$SAD_i = \sum_{\langle x,y\rangle \in M} \|I_{k+1}(x + dx_i, y + dy_i) - I_k(x, y)\|$$

Then, if we take an index j for which $SAD_j \leq SAD_i \ \ \forall i = 1 \dots n$, then $\langle dx_j, dy_j\rangle$ is the offset of the baby's face between the two images. This allows us to track the face from image to image.

Use of template matching for cut-in detection

In our application, we don't need template matching for tracking ; for this we have several other algorithms which give good results. Our use of template matching will be *to detect objects which hide the Stop&Go target*. The basic algorithm, which is a little different than the classic one, uses a real-valued mask instead of a binary one. The algorithm is described here and is being started each time a new Stop&Go target is found :

1 Constants : W, H - the width and height of the template. In our implementation we used $W = 60$ and $H = 45$.

2 Input :
- an input image I_k.
- a rectangle $\langle x_1, x_2, y_1, y_2\rangle$ donating the rear projected rectangle[4] of the Stop&Go target on the image. Our goal is to determine if the content of this rectangle (on the image) has dramatically changed relative to previous images.
- a real-valued mask $M_{i,j} \ \ i = 1 \dots W, j = 1 \dots H$, initialized to 0.5 at the beginning. The cell $M_{i,j}$ is the confidence that this point in the template belongs indeed to the tracked object and not to the background.

[4] the projection of the rear of the car ; see subsection 6.2.1

- a template $T_{i,j}$ $i = 1 \ldots W, j = 1 \ldots H$. $T_{i,j}$ is actually the value that exists in location $\langle i, j \rangle$. If $M_{i,j} = 0$, it's meaningless.

3 Perform warping from the image I_k in the locations x_1, x_2, y_1, y_2 to an auxiliary image AUX of width W and height H.

4 If this is the first time, copy AUX to T and continue to the next image.

5 Otherwise, calculate the sum of absolute difference with the existing template, taking into account the confidence :

$$SAD = \sum_{i=1..W, j=1..H} M_{i,j} \| T_{i,j} - I_k(x_1 + i - 1, y_1 + j - 1) \|$$

6 Update the confidence accordingly.

Each time the SAD is higher than a given threshold, the system concludes that something is found between us and the Stop&Go target. The result is an immediate disconnection of the system.

6.5 Results

In this section we demonstrate, by showing real-life situations, that the particle filtering mechanism optimally exploits the analysis coming from the five algorithms.

6.5.1 Typical examples of the particles filtering

Figure 6.29 shows a typical daylight situation. A car is found about 30 meters in front of the host vehicle, and is initialized by the shadow detector. The target is quickly validated by all the algorithms except the motion detector (since the projected image of the car is not moving).

In figure 6.30 we see a continuation of that sequence with a car entering from the right ("cut-in" situation). The car is initialized very early by the shadow or motion detection (whichever detects first ; usually in the second image after the appearance) and is maintained at the beginning mainly by the shadow detection. The figure shows the scene a little bit later, after the rear of the car is beginning to appear. At this stage we can already see some initial contributions from the lights and vertical edges detection. As seen, motion detection is giving a high probability to the existence of the target, but very little localization information.

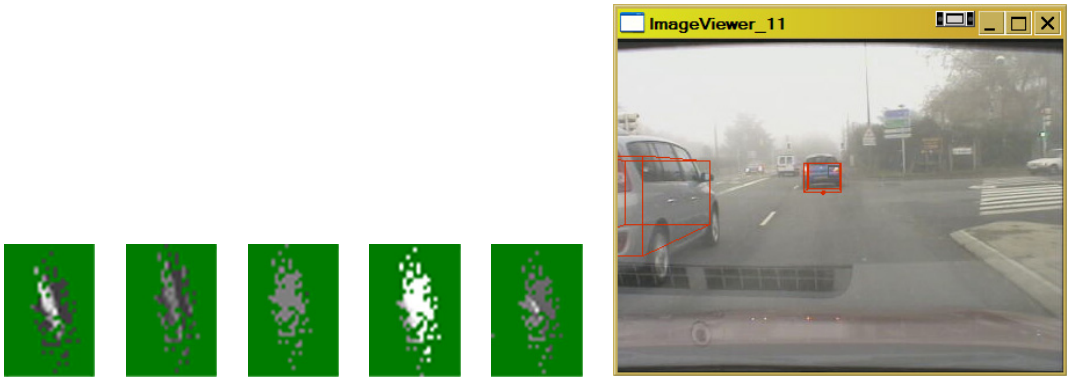
6.5.2 Analysis on a long sequence

We tried our system on a long sequence (approx. two hours) taken in the wide parisian area in France. The sequence contains highway scenes, urban scenes, tunnels and bridges, and a wide variety of lighting conditions (daylight, darkness, "afternoon conditions" with direct sun and long shadows etc). We hereby bring the results

Figure 6.31 bring several examples for successful detection. We see that the system copes well, in these cases, with all types of lighting conditions.

(a) (b) (c) (d) (e) (f)

FIG. 6.29 – A typical situation of a vehicle in front of our car. The frontal view (f) and likelihood functions (from bird's eye view) of the shadow detection (a), the vertical edges detector (b), the symmetry detector (c), the motion estimator (d) and the rear lights detector (e). Values are denoted from black (0) to white (1).



(a) (b) (c) (d) (e) (f)

FIG. 6.30 – A vehicle entering from the left ("cut-in" situation). The frontal view (f) and likelihood functions (from bird's eye view) of the shadow detection (a), the vertical edges detector (b), the symmetry detector (c), the motion estimator (d) and the rear lights detector (e). Values are denoted from black (0) to white (1).

FIG. 6.31 – Successful detections in a variety of conditions.

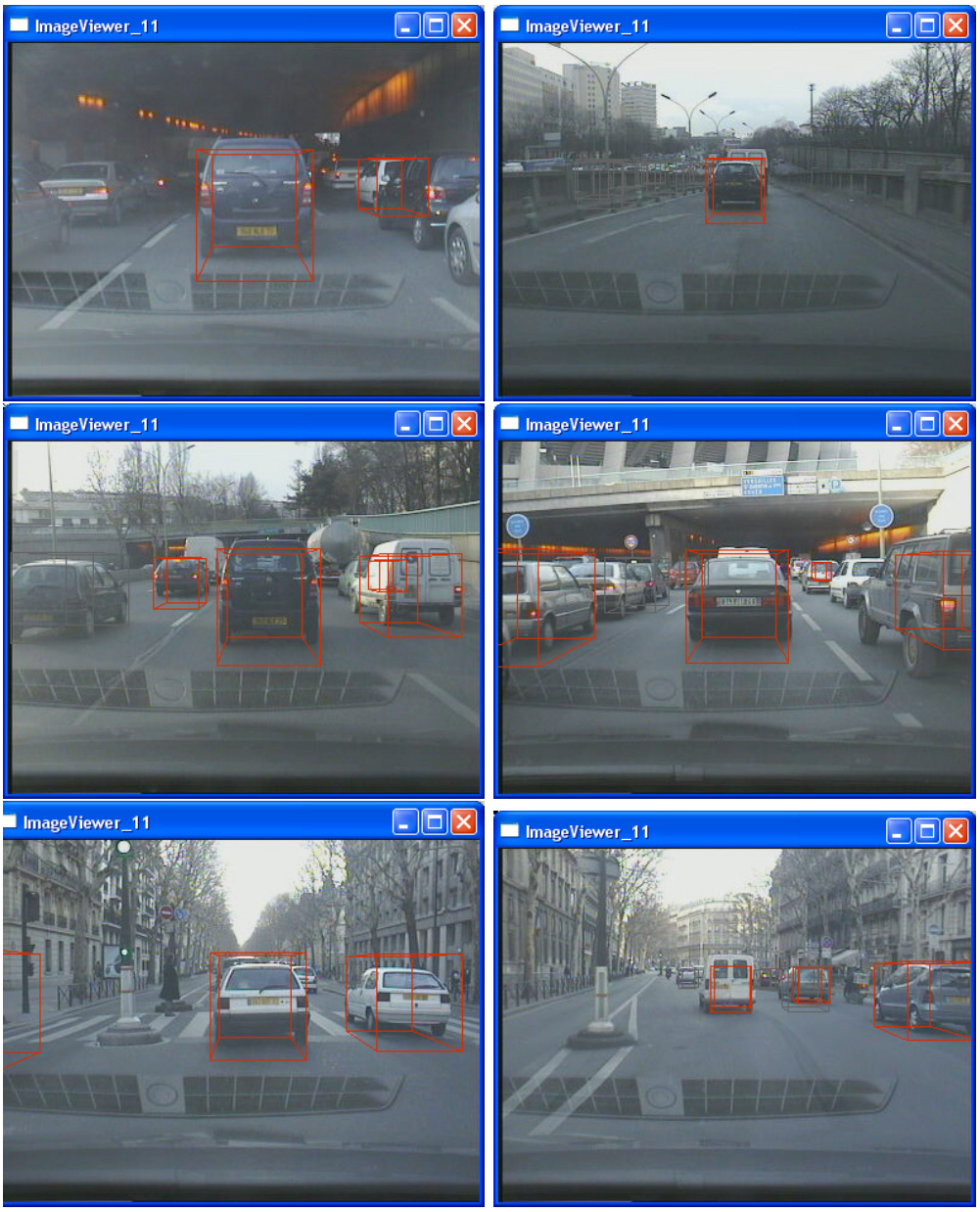


FIG. 6.32 – Successful detections in a variety of conditions.

FIG. 6.33 – Case 1 (see text).

The more interesting cases, of course, are the non-successful ones... We will now analyze some important examples and conclude with the identified weaknesses of the system.

Case 1

This case is an example of how bad lighting conditions can influence the shadow detection algorithm. In figure 6.33 we see a target entering the scene, but due to the direct sun light the shadow created is larger than the car. The target is detected by the motion detector but its localization, which is supposed to be done according to the shadows, is not correct. We see that several images later the target is still not localized.

The problem occurs at an early stage of the car localization, when symmetry and vertical edges are still not effective. When they are, the target is too far from the actual car.

Case 2

This case (see figure 6.34) is not entirely different from the previous one. Here, it is the *direction* of the light which causes a long shadow to the car. This time the shadow is really of the car - but it's too long. As a result, a target is misplaced.

Again, the problem occurs at an early stage of the car localization, when symmetry and vertical edges are still not effective. When they are, the target is too far from the actual car.

FIG. 6.34 – Case 2 (see text).

Case 3

This case shows the results of the weak signals while driving in a tunnel. Two of the most important algorithms - shadow and real lights - do not function well. We see in figure 6.35 that shadows barely exist and that lights cannot be detected because the background of the tunnel is all colored in yellow and red. As a result, the load is falling mainly on the symmetry algorithm, since vertical edges are also weak in the absence of light. The symmetry algorithm itself, however, cannot hold the target in place for a long time.

Case 4

Here we see another aspect of the tunnel problem. In figure 6.36 we can see that the shadow algorithm believes to have found a good shadow, and it gives the likelihood according to it. However, the position of the shadow is wrong, and the target is mal positioned. At the beginning, the symmetry and vertical edges algorithms are holding the target in place (rear lights are not red in the environment of the tunnel). But at some point, the shadows algorithm gives high confidence for the false localization.

Case 5

This is an example of a car, which is very close to us, but is not initializing a target. As seen in figure 6.37, the shadow algorithm does not detect a correct shadow and therefore cannot initialize, and the lights algorithm cannot find a high value in the V plane, since the lights are not red at all.

Case 6

In figure 6.37 we see a case of a major false detection. The target is initialized by red-yellow color coming from the afternoon sunlight, creating by coincidence a lights-and-plate structure which initializes a target. The target later is supported by the same "rear lights", and no other algorithm can decline this target : shadow algorithm is not active because the

FIG. 6.35 – Case 3 (see text).

FIG. 6.36 – Case 4 (see text).

FIG. 6.37 – Case 5 (see text).

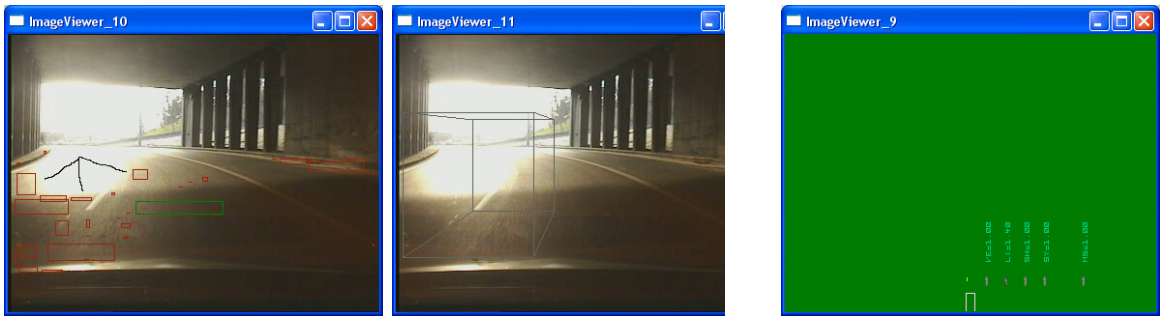


FIG. 6.38 – Case 6 (see text).

road-rectangle is not providing a clean histogram. The symmetry and vertical edges are not active because the target is too close to the right side of the image.

6.6 Conclusion

In this chapter, we have presented an application for keeping a steady distance between our car and the car in front. The application, known as "Stop&Go" or "low-speed ACC", is using vision and lidar to find out the headway between us and the car in front. This information is used to control the car's braking and acceleration pedals and therefore the car's speed. We have presented the various algorithms used to analyze the vision signal coming from the camera as well as the lidar signal.

We have tested the operation of the system on a long sequence. The application seem to work fine in most conditions. Some adjustments need to be done in specific cases.

Chapitre 7

Application II : pedestrian detection and impact prediction

Detection of pedestrians from a video camera installed on a moving vehicle is an important problem with applications in driver assistance systems. This domain has attracted a considerable amount of research in recent years [Tsuk 85][Leun 87][IHar 98][Bert 02]. However, the task of creating a reliable system which predicts at time k the probability of an impact with a pedestrian at time $k + \delta$ received less attention so far. Generally, the focus in pedestrian detection is to achieve high rate of detection versus false detections. While this effort is preserved here, there is an additional focus on exact localization of the pedestrian and efficient estimation of its speed and direction.

7.1 SEVILLE-based applications

This section describes an application for pedestrian detection and impact-prediction using only the SEVILLE method. In the next sections, we will describe more advanced applications using particles filter and medium level algorithms.

The application predicts, at time k, the probability of impact between our vehicle and a pedestrian at time $k + \delta$.

Our system runs on the RT-MAPS framework [Steu 00]. Figure 7.1 presents the diagram of the system : the input image, after being converted to a gray intensity image, enters the SEVILLE component, a pedestrian detector which uses a binary classifier to detect all the pedestrians in the image. These detections - in image coordinates - are transferred to the *impact predictor*, which is interpreting these results in terms of world coordinates (according to camera parameters), tracks the different targets and predicts while calculating their speed and direction. The output of the impact predictor is double :

First, it outputs the same detections it received as input, while coloring the "dangerous" pedestrians in red (see figure 7.2). These detections are drawn on the input image using an overlay drawing component, and presented to the user with an image viewer.

Second output of the predictor is the *impact probability*. This probability is estimated according to a predefined pedestrian model.

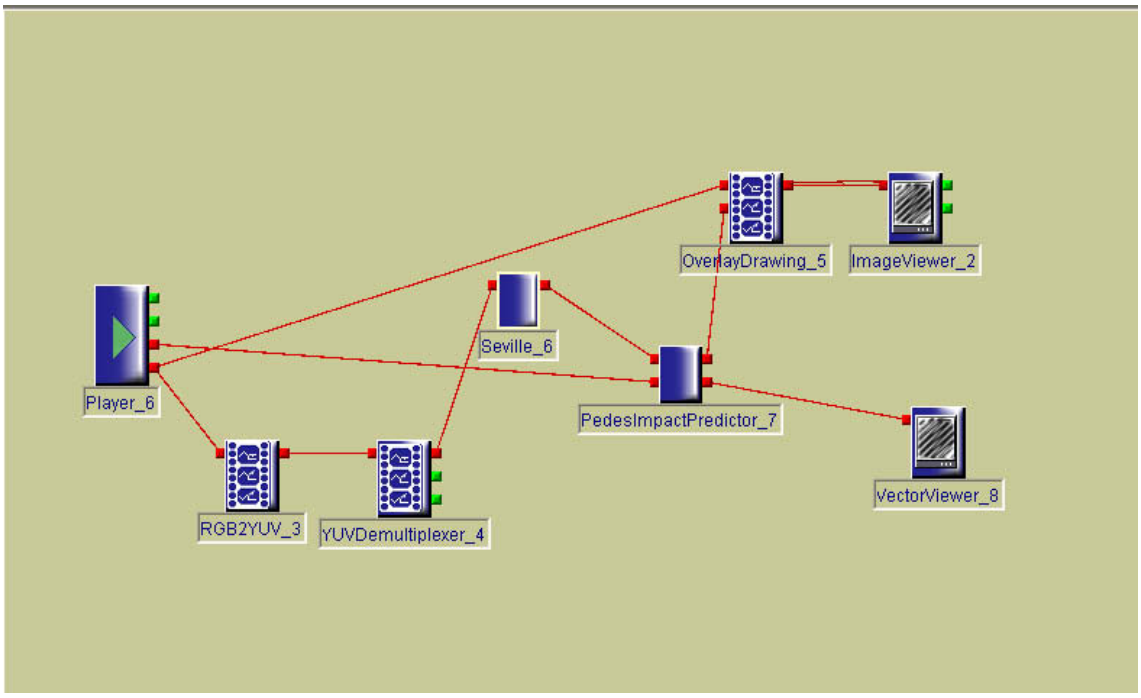


FIG. 7.1 – The RT-MAPS diagram used in the application.

As we will now see, the pedestrian detector is not just analyzing static images. It has a built-in simple temporal filter which performs some minimal tracking on the pedestrians. Thus, the detections which are transferred to the impact predictor already contain an ID which might link them with those of the previous image.

7.1.1 The SEVILLE pedestrian detector

SEVILLE (SEmi automatic VIsuaL LEarning) is described in section 3.7 as a system for fast collection of training data. The same system was integrated into an RT-MAPS component and it detects pedestrians on gray images. The detection procedure is performed in the same way : given an input image, we apply an AdaBoost classifier to consecutively classify all the sub-windows in the image. Here we used the same configuration of 24x48 pedestrians described in section 3.7, and the system examines about 400,000 sub-windows of each image, consisting on all the possible sub-windows in 16 scales, starting with the basic size of 24x48 and enlarging the size in 10 percent each time (26x53,29x58 etc).

The AdaBoost classifier is using the same illumination independent features described in section 3.3. The classifier used is the one obtained at the last step of the SEVILLE collection procedure.

The AdaBoost detections are passed to further processing - grouping and temporal filtering. These two units will be discussed now.

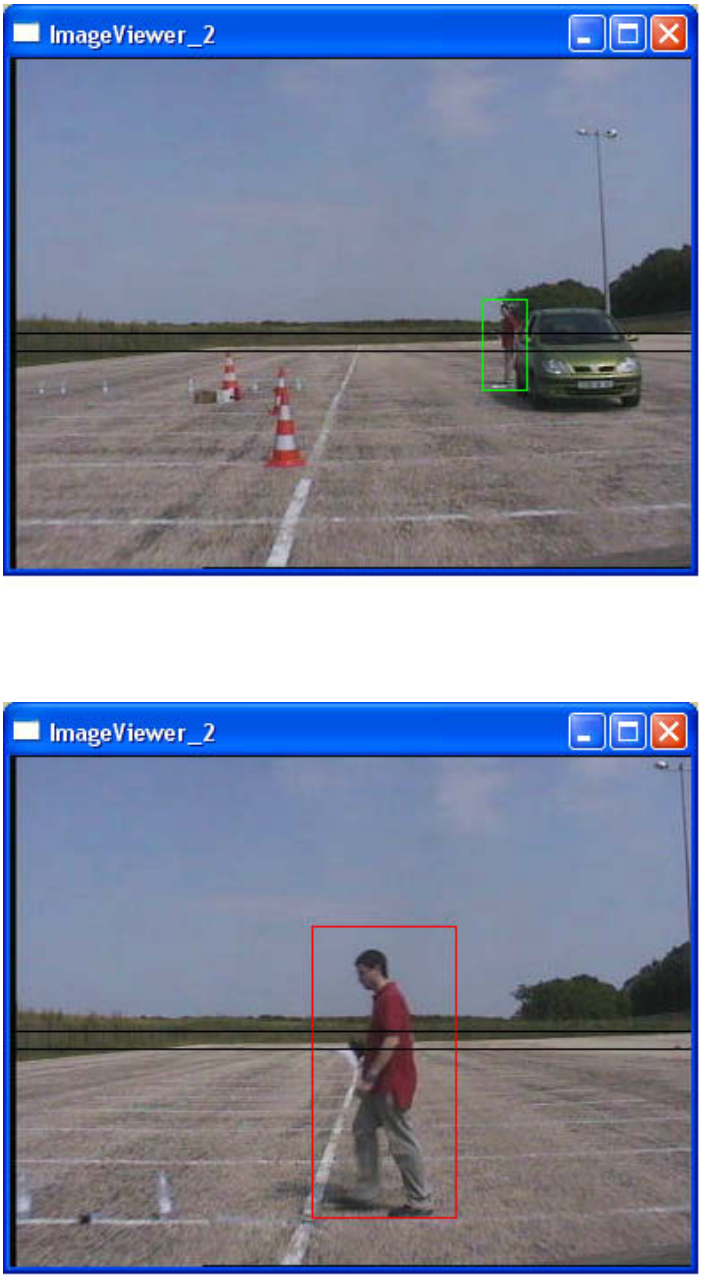


FIG. 7.2 – A pedestrian detected in an early stage (top) and when it is in risk of impact (bottom).

Grouping adjacent detections

The detection process is usually yielding several rectangles around a detected pedestrian (see figure 3.40). To remove this effect, we have implemented grouping of adjacent rectangles.

The detection process provides us with all the rectangles which received a score of more than a certain threshold λ. First, we divide these detections into sets. Two rectangles belong to the same set if they overlap (we take the transitive closure of this relation ; that is, is A overlaps B and B overlaps C, then A and C are also in the same set).

Each set S yields one final detection R, which is the weighted average of all the rectangles. The weights are the scores :

$$left(R) = \frac{\sum_{r \in S} left(r)(score(r) - \lambda)}{\sum_{r \in S}(score(r) - \lambda)}$$

$$right(R) = \frac{\sum_{r \in S} right(r)(score(r) - \lambda)}{\sum_{r \in S}(score(r) - \lambda)}$$

$$top(R) = \frac{\sum_{r \in S} top(r)(score(r) - \lambda)}{\sum_{r \in S}(score(r) - \lambda)}$$

$$bottom(R) = \frac{\sum_{r \in S} bottom(r)(score(r) - \lambda)}{\sum_{r \in S}(score(r) - \lambda)}$$

The temporal filter

The temporal filter contained in the SEVILLE component is a simple one which allows (i) to remove contemporary false detections and (ii) to perform minimal tracking in the sense that the detections passed out of the component already contain an ID which is consistent in time.

The filter is based on the notion of *confidence*. Each detection has a confidence which increases when the detection repeats itself in consecutive images and decreases otherwise.

The algorithm maintains, at all times, a set $\mathcal{D}$ of detections with confidences. In addition to the confidence, each detection can be "visible" or "invisible". In the output of the temporal filter, the systems draws only the "visible" detections.

The algorithm has four parameters : $\underline{C}$, $\overline{C}$, $C\uparrow$ and $C\downarrow$. Upon receiving fresh detections $d_1 \ldots d_n$ from the current image (after having been grouped as described above), the following operations are performed :

- For each $d_i, i = 1 \ldots n$, check if there is an existing detection $d \in \mathcal{D}$ such that d and d_i are *near*. The definition of *near* is the heart of this filter and can be done in several ways, which will be discussed further.
- If there is such detection, copy the location of d_i to d (that is $left(d) \leftarrow left(d_i)$, and the same for right, bottom and top). Increase the confidence of d by one. Do not increase the confidence if it reached a maximum of $\overline{C}$. If the confidence is superior of $C\uparrow$, the detection becomes "visible".

- If there isn't such detection, add d_i to the set $\mathcal{D}$ with an initial confidence of $\underline{C}$, and mark it as "invisible".
- For each $d \in \mathcal{D}$ which was not coupled with a new detection (and which is not one of the newly added detections), decrease the confidence by one. If it reached 0, remove it from $\mathcal{D}$. If it dropped below $C\downarrow$, make it "invisible".

In our implementation, typical values for the parameters were $\underline{C} = 2$, $\overline{C} = 10$, $C\uparrow = 5$ and $C\downarrow = 4$.

7.2 Particle-filter-based applications

In this section, we present a more advanced application, using particle filter. The use of a filter allows us to run in real time while preserving the same detection results.

7.2.1 Introduction

The Pedestrian Detection Application uses the same general framework as the Low Speed Obstacle Detection application : we apply a set of MLAs (Medium Level Algorithms) and fuse the results through a particle filter. The fusion approach allows us to get the best from algorithms that give weak results when taken individually.

The selected MLAs themselves are rather classical (legs/diagonal detection, body detection, vertical edge detection and motion estimation) - see for example [Bert 02] - except for the pedestrian classifier, based on an innovative hardware-friendly AdaBoost/GA algorithm.

Globally, our particle filter based approach can be considered as very close to the state of the art in the domain of visual pedestrian detection [Bert 02], though most of the existing systems use stereo vision instead of monocular vision [Gavr 01].

7.2.2 Pedestrian representation

When we speak about a location of a pedestrian on the world, we will specify a pair of the form $\langle x, y\rangle$. Unless stated otherwise, we denote by this a pedestrian which stands on the ground with its bottom center part being at the point $\langle x, y, 0\rangle$, as drawn in Figure 7.3.

7.2.3 The basic algorithms and the likelihood function

When speaking about image processing algorithms in this context, we are referring to a type of algorithms which is being run in the following way :

- Pre-processing : one time per image of the video sequence. The output of this stage is a structure which contains the pre-processing results (shadows, symmetry functions etc.)
- Likelihood function : can be run many times in each image. This function is using the preprocessing results to evaluate the probability that a given target exists in a given location.
- Create new targets (only a part of the algorithms) : this step is using the pre-processing data to create new targets. Figure 7.4 explains this method of work.

FIG. 7.3 – The box of a pedestrian, and its origin in the bottom centre point.

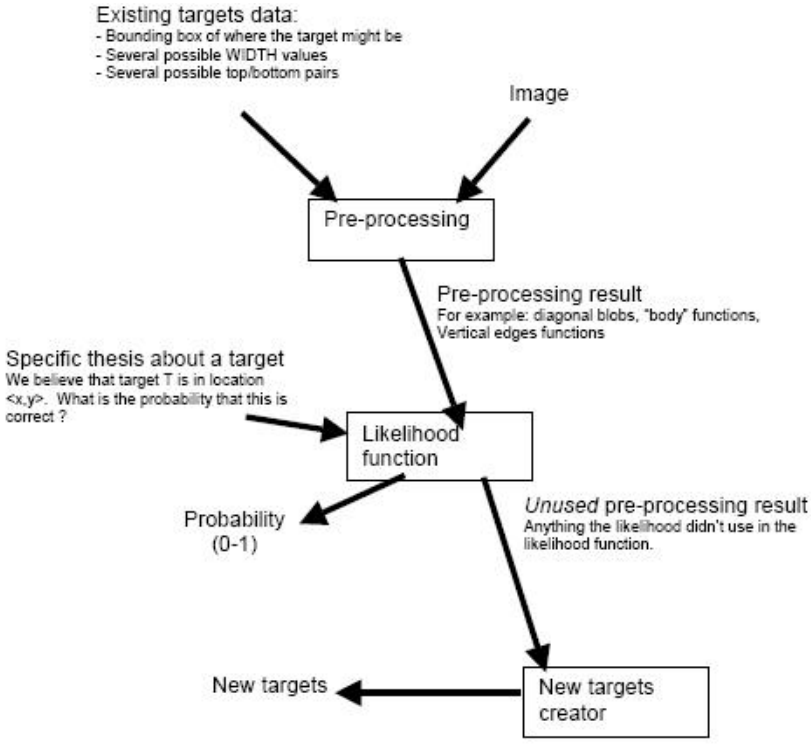


FIG. 7.4 – The steps of the particle filtering process.

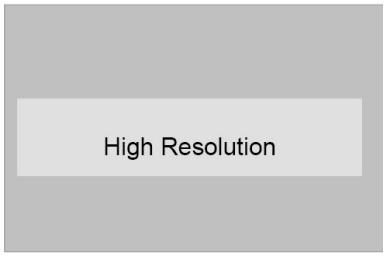


FIG. 7.5 – The region of interest of the input images.

7.2.4 The input of the algorithms

Typical algorithm will get as input the image in high resolution. This is a gray image, originated from the Y plane of an input YUV image. The dimensions of the image are 384x288 (half PAL)1 with a ROI (region of interest) defined as the rectangle $\langle(8, 100) - (376, 188)\rangle$.

Figure 7.5 shows the location of the ROI on the image. This ROI was selected so as to detect a pedestrian δ before impact. Taking a bigger ROI is not necessary : a pedestrian bigger than the ROI is too much next to the car.

In addition to the images, an algorithm will typically get some information about the previously detected targets, so it will know to focus the pre-processing to the areas where we are likely to find targets. The information about an existing target consists of :

- a bounding box that covers all the area where we believe the target to be ;
- several values of WIDTH that estimate what could be the width of the target ;
- several pairs of values <top,bottom> that represent some possible values for top and bottom of the pedestrian, according to previous estimations.

7.2.5 The projection and retro-projection

Image processing algorithms work on the screen ; real targets exist in the real world. These are different coordinates, which need converting to both directions. In our application, we assume specific parameters of the camera, with which the video sequence was taken. Using the values of these parameters, we can provide a projection of a point $\langle x, y, z\rangle$ to a screen point $\langle u, v\rangle$ and vice versa. Note that throughout this document, when we say that we project a pedestrian located at $\langle x, y\rangle$ to the screen, we mean that we do the following sequence of operations :

- produce the 3D box of the pedestrian, which is located at $\langle x, y\rangle$ (as defined above) ;
- project only the closer 4 points of this box ;
- calculate on the screen the projection of the rectangle.

7.2.6 The particle filtering framework

The pedestrian application uses the same known framework of particle filtering that is used by CAMELLIA's low speed obstacle detection application, and by the face tracking application. The system uses the input of 5 MLAs (Medium Level Algorithms) :

- Legs detection,

FIG. 7.6 – The left diagonals in the image are marked with a purple rectangle, while the right diagonals are marked with yellow ones. The intersection between the two is marked by a pink triangle - and this exists only in the area of the legs of the pedestrian.

- Body detection,
- Vertical edges excluding algorithm,
- Motion segmentation,
- Image Features (AdaBoost/GA).

Only the legs detection and motion segmentation algorithms are producing new targets, while all the 5 algorithms are supporting existing ones (via the likelihood function mechanism).

7.3 The medium-level algorithms (MLA)

7.3.1 The legs/diagonal detection algorithm

Introduction

The legs detection, or the diagonals detection, is a special algorithm that was designed specially to detect and to verify the existence of pedestrian in a video image. The principle behind this algorithm is that, usually, the legs of a walking pedestrian are found in diagonal positions, in an alternating fashion. Figure 7.6 shows a typical scene, while marking the places in the scene where left and right diagonals are found. We can see that the only place in the image where there are both directions of diagonals is the location of the legs of the pedestrian.

Given the assumption that intersection of left and right diagonals are occurring mostly in the area of pedestrians' legs, we run some low level processing of the image in order to detect such areas, as described in the next subsection.

However, if we stick only to the assumption above, we are likely to be disappointed in many scenes which contain a walking pedestrian. This is because, usually, we indeed have both direction of diagonals in the area of the legs, but often not in the same image. In many cases the pedestrian's legs area contain a left diagonal, and the right diagonal appears only several images later. To catch situations like this, we have added a temporal filter to

FIG. 7.7 – The gradient image.

our algorithm. Roughly speaking, each detection of a diagonal in the images "lives" several images, and not just in the image it was detected in. If an opposite diagonal appears several images after, than an intersection occurs with the "old" diagonal. This is formally described in the next subsection.

Algorithm first step : detecting the edges image

The first step in detecting diagonals is to generate the image of the edges of the image, in order to further find the diagonal ones. For this we calculate a dilated version of the image, named D, and an eroded version of the image, named E. Both erosion and dilatation are done with the 3x3 element $((1,1,1),(1,1,1)(1,1,1))$. The edges image is obtained by calculating the image (D-E) - that is, the difference between the dilated image and the eroded one, as shown in Figure 8. Note that all the operations are done only on the region of interest of the image as described above.

Why does the image (D-E) represents the image of the edges of the figure ? This can be intuitively explained as follows :

When dilating with a 3x3 element as specified above, each pixel gets the maximum value of its 3x3 environment. When eroding with this element, each pixel gets the minimum of this environment. Therefore, each pixel in the image (D-E) contains the difference between the maximum and the minimum of its 3x3 environment. This difference is high in pixels, which reside on the edges of objects - they have a large difference between the values of the object, which are in their 3x3 environment, and the background, which is also in that environment. This is in contrast with pixels which reside in the interior of objects, or totally on the background, which have a relatively homogenous 3x3 environment. Therefore, in the image (D-E) we see high values where we have edges in the original image.

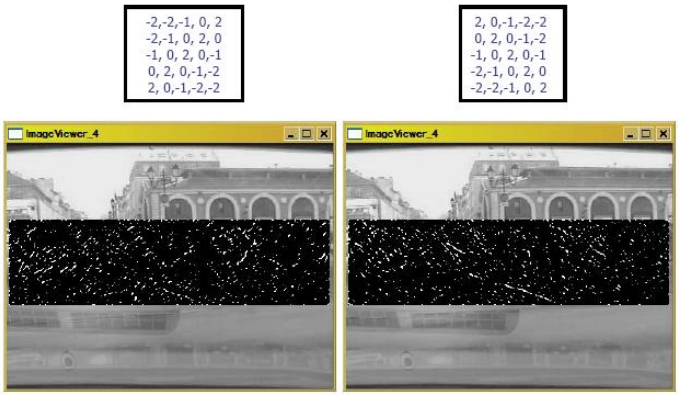


FIG. 7.8 – The result of applying linear filtering with "diagonal" elements : the image on the left is the result of linear filtering with the element above it, and the same on the right.

Creating the diagonals image

Once we have the edges image, we want to highlight the right and left diagonals (each direction in a different image). To do this, we run a linear filtering of this edges image with one of the 5x5 elements described in Figure 7.8. The results of applying these linear filters are also demonstrated in Figure 7.8.

Eroding again to sharpen the detection

Many times, even after the linear filtering, we find that the resulting image still has some traces of edges which are not in the correct diagonal direction, as seen in Figure 7.9. By applying erosion with a sharp 5x5 diagonal element we leave only the diagonals which we really want.

Detecting the diagonals

All the operations until now gave us two images, one emphasizing the right diagonals and the other emphasizing the left ones. To get a discrete detection of such diagonal edges we now perform a simple threshold of the two images, followed by labeling and blob[1] analysis. From the resulting blobs, we take only the blobs which are larger than a specific value. The important characteristic of a blob for us is its diagonal length, because this is actually the length of the detected diagonal edges. Therefore we go blob by blob and compare its diagonal length to a fixed value, L, which was empirically defined as $\sqrt{150} \sim 12.24$.

In practice we do the comparison by taking the square of the length of the blob, adding it to the square of the width of the blob, and checking if it exceeds 150.

A detection of such a blob means that we found a diagonal in the respective direction. Figure 7.6 shows a typical scene with some detected diagonals. The yellow and purple squares are the bounding boxes of the detected right and left blobs, respectively.

[1] A blob, in English, is defined as a "soft, amorphous mass". In computer vision we mean an object in an image that does not have a specific form.

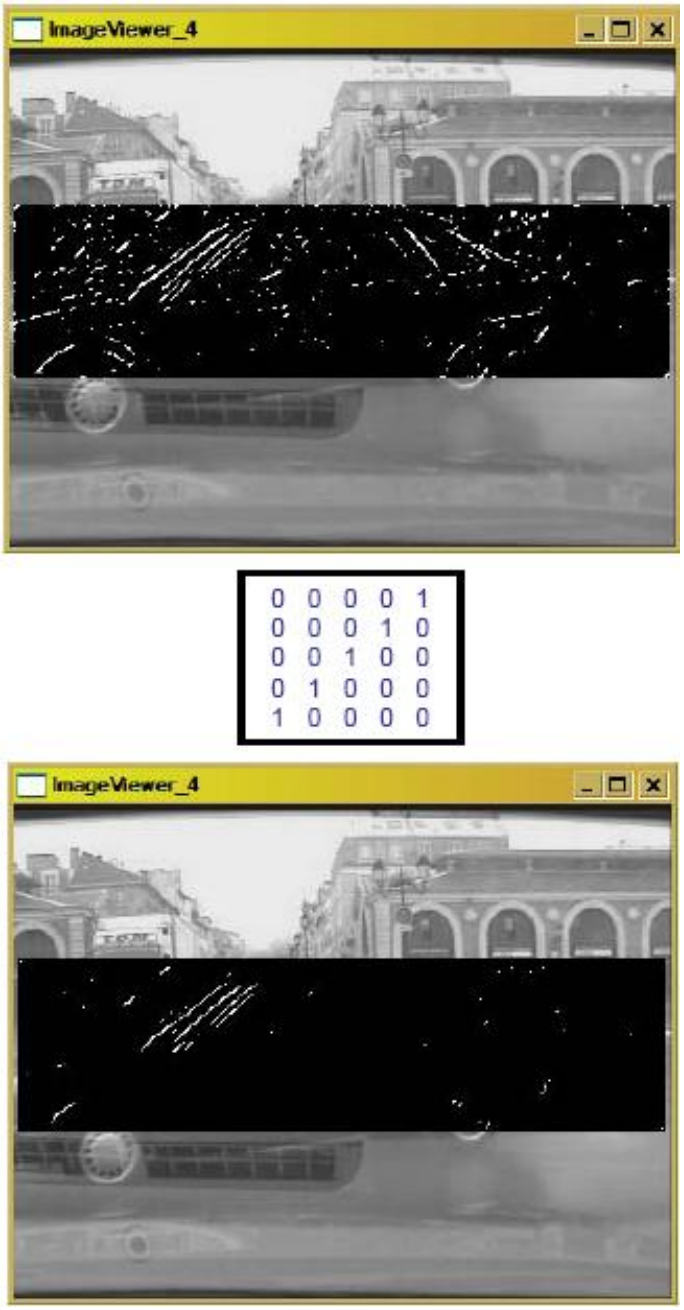


FIG. 7.9 – The result of linear filtering with the left diagonals (upper image) still contains many right diagonals, due to the strong edges to the closely-passing car. After erosion with a simple 5x5 diagonal element (in the middle), we get a better image without right diagonals (lower image).

The temporal filter

As mentioned in the introduction, our goal is not just to find an intersection of left and right diagonals in a given image, but to find even an intersection of such two diagonals that appear with a temporal offset of several images. The number of images chosen for this filter was 15, because this is generally the average duration of a pedestrian step. Therefore we proceed as follows :

1 Each detection of a blob, as described in the previous subsection, creates a "diagonal object" with age of 0 and with the relevant direction (right or left).

2 Each image we go over all the objects and increase their age by 1. Objects which are older than 15 are deleted.

3 On the remaining objects we run a process, which detects all the intersections between a left object and a right object. Each one of these intersections is described by the intersection of the squares of the two blobs, and its age is the minimum of their ages.

The resulting set of intersections are the places where we are likely to find a pedestrian's legs, and therefore it forms the final output of the processing of this algorithm. All the rest of the work is done in the higher-level code (the likelihood function and the generation of new targets).

The likelihood function

The likelihood function is designed in view of the following principles :
- A result coming out from the processing step of the algorithm, is likely to be a place of legs of a pedestrian.
- The more the age of the result, the less it is exact.

Therefore we have designed the following function :
- For a given particle, find a result of the algorithm, which is the closest to the projection of the particle's pedestrian.
- If the distance to this result is too large then we conclude that this pedestrian is not supported by diagonals, and we give a likelihood of 0.25 (which is usually the minimum likelihood of the MLAs). But what is "too large" ? This depends on the age of the detection. The oldest the detection, the larger its span is. That is, if we have a diagonal detection of 10 images ago, it will support particles which are in a wider range, because these might be pedestrians that have already moved. In practice we use the value of AGE*3 as the number of pixels which are considered as "too large".
- If the pedestrian was found in the range of the detection, we give a likelihood function which is higher as the age of the detection is smaller - this is because a fresh detection of diagonal from the current image is more likely to imply the existence of a pedestrian than a detection from 10 images ago. The practical value is 1.0 - (0.025*AGE).

Creation of new targets

Creating new targets is done simply when some diagonal detection was not used to support an existing pedestrian. In such case, we initiate a new pedestrian target with particles spread

FIG. 7.10 – Several examples of a pedestrian.

around the projection of the detection. This is similar to what is done in shadow detection in the LSOD application.

7.3.2 Learning algorithm using AdaBoost

This algorithm uses the learning technique from chapter 3 to find a good classifier of pedestrians. Each simple feature is a small "classifier" that knows to decide, upon receiving an image rectangle, if this rectangle contains a pedestrian (we will later elaborate how it does it). AdaBoost selects a set of these features and gives each of them a weight. As explained in chapter 3, this set is a "voting system" : upon receiving an image rectangle, each classifier gives its opinion, and if the total weight of the ones which answered positively exceeds 0.5, it's a pedestrian.

As explained in chapter 3, the algorithm described here is not used, in our context, for pedestrian detection, but for classification of a given image rectangle : it tells us if a given rectangle is aligned on a pedestrian. Figure 7.10 shows several examples of positive and negative cases.

The classifier used is the one described in section 3.4. The only thing which is new is the likelihood function. The likelihood function simply returns 1 if AdaBoost responds "yes", and 0.1 if it responds "no".

7.3.3 Body detection

Introduction

The body detection algorithm is another algorithm, which is designed specially to support the existence of pedestrians. The algorithm is based on the fact that in many cases the upper part of the body of a pedestrian forms a relatively homogenous "stain" on the image, while the borders of this part contain some edges (on the right and left - the border of the pedestrian ; on the top - the neck and head ; at the bottom - the difference between the shirt and the trousers, and the legs).

To detect such effect, we use the same edges image used in the previous algorithm. Figure 7.11 shows an example of how a typical pedestrian looks in such an image.

Algorithm steps

The course of the algorithm begins by producing the edges image as described above. There, we run a sequence of some measures calculations. We begin by two small ROIs and calculate the sum of the pixels in the large ROI, minus the sum of the pixels in the small ROI.

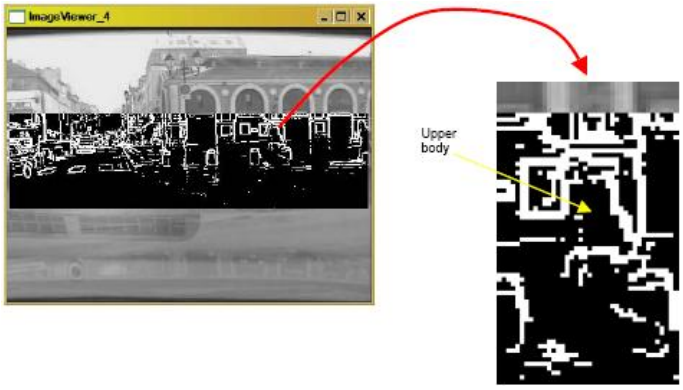


FIG. 7.11 – A typical pedestrian in an edges image. In the zoomed part on the right we see that indeed the upper body of the pedestrian is characterized by a "hole" block whereas the outline of this body part has some edges.

FIG. 7.12 – The moving pair of ROIs in the body detection algorithm.

Note that this value will be the highest in a case of a pedestrian upper body as described above. We proceed by dividing this value by the width of the large ROI, to be independent of the size of the pedestrian.

After we do this calculation for these two ROIs, we move both of the ROIs one pixel to the right and calculate again. In this way we create a one dimensional function drawn in green dots in Figure 7.12.

Note that the highest value of this function is, indeed, in the center of the pedestrian body.

This algorithm is used only for supporting existing targets, using the likelihood function. Creating of new targets is not done with it, because it sometimes gives high values to places where there are no pedestrians, as seen also in Figure 7.13.

The likelihood function

The likelihood function is directly derived from the output function of the algorithm. Given a pedestrian in position $\langle x, y, 0 \rangle$, we project it to the screen and take the center

FIG. 7.13 – The resulting function of the body detection algorithm. The green dots express, for each horizontal value, the extent of this function. The lower the point, the higher the value is. We see that indeed in the center of the pedestrian's body, the value is the largest. However, there is another area of high values about 20 pixels left of that location.

horizontal value of the projected rectangle. We take the value of the output function in this horizontal location. This value is typically large. We divide it therefore by a constant value (practical value stands on 60). The result is the likelihood value between 0 to 1.

7.3.4 Motion detection

Introduction

The motion segmentation (or estimation) system was already used in the first two applications of CAMELLIA and is widely described in a separate chapter. We therefore describe here only the use of this system as another MLA in our application.

The motion segmentation is, as it was in the LSOD application, highly useful to detect insertions of new objects to the scene. Inserted objects are, by definition, moving, and this algorithm can give an unmatched capability of spotting such objects. In this application the motion segmentation helps even more, because the ability of all the other algorithms to detect any type of pedestrian, in any location and background, is obviously limited.

The likelihood function

The motion segmentation algorithm is being run on the images. When an object was detected, as shown in Figure 7.14, we have a rectangle on the screen. The likelihood function is obtained by projecting the particle's pedestrian to obtain a projected rectangle, and, similarly to the corresponding likelihood function in the LSOD application, calculating the intersection rectangle between the two rectangles. The likelihood (values 0-1) is obtained by the following

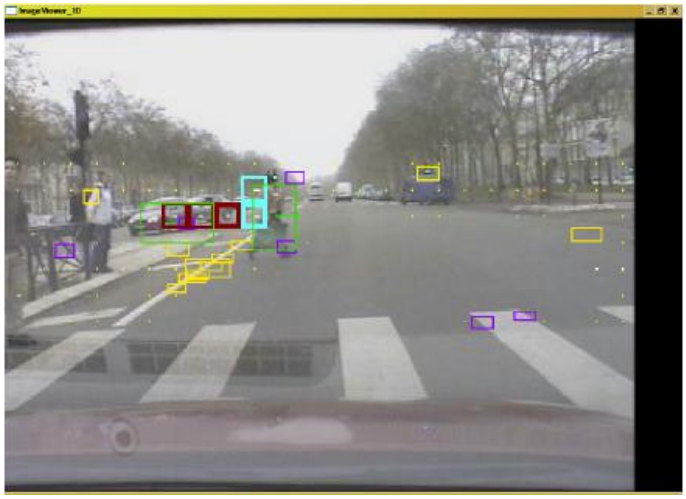


FIG. 7.14 – A pedestrian's movement is detected (green rectangle with an arrow from its middle to the right). Note that also the car on his left is detected by its movement. Here we rely on the other algorithms to understand that this moving object is not a pedestrian.

division :

$$\frac{Surfaceofintersectionrectangle}{SurfaceofMSobjectrectangle}$$

As in the LSOD application, we do not take into account the size of the pedestrian rectangle. The same explanation of this decision holds here.

7.3.5 Vertical edges excluding detection

Introduction

Vertical edges in its original form (like in the LSOD application) would not work in this application. It's enough to look at Figure 7.15 to understand that a typical scene can contain many vertical edges which are much stronger than the pedestrian. Moreover, a typical pedestrian will never have very strong vertical edges, due to its moving legs, hands structure and the head. This is why we decided to make an excluding algorithm : an algorithm that will say where a pedestrian is not found. A pedestrian is not found in areas of strong edges. This algorithm is based on computing the vertical sums of the vertical edges image, as in the LSOD application, but the likelihood function will give high likelihood only when the value is under some fixed number.

The basic algorithm

The algorithm runs as follows :

– produce the vertical edges image by filtering the original image by the filter described in Figure 7.15 ;

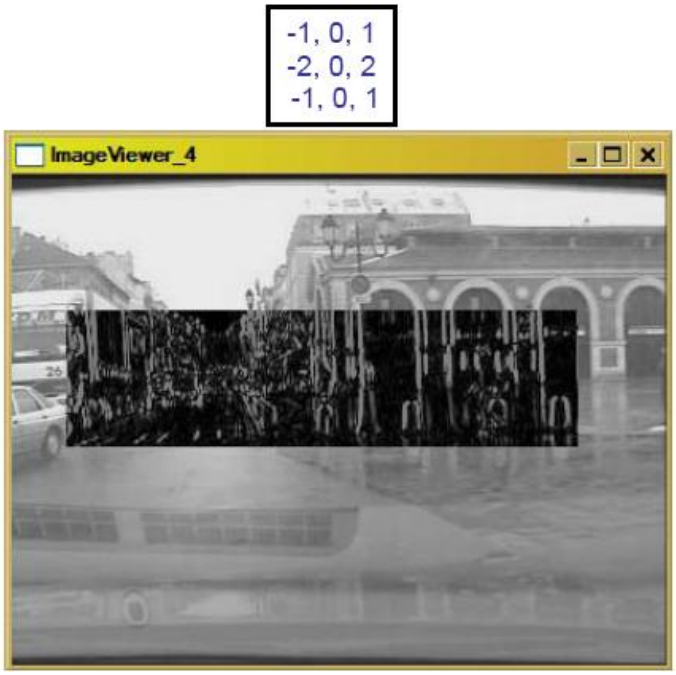


FIG. 7.15 – Vertical edges image. The pedestrian is on the right part of the region. Above the image, the 3x3 filter used to produce this image.

– for each input existing target's bounding box B, output a vertical summing sequence $V_1 \dots V_n$ T of integers that describes the vertical extensiveness of the image at each point.

Figure 7.15 is a good example for a typical vertical edges image of a scene with a pedestrian. Note that indeed the pedestrian is not giving very strong lines.

The likelihood function

The likelihood function starts as always by taking a particle and producing the projected rectangle of the pedestrian. After that, we check the maximum value of the vertical summing in the area of this rectangle. If the value exceeds a fixed number (actually 70) then the likelihood is 0.25 (which is very low). Otherwise the likelihood is 1.

7.4 Impact prediction

7.4.1 Introduction

The application we present here has two outputs. The first is the raw detections of pedestrians, presented in the previous sections. The second is a single real number - the probability that an impact with a pedestrian will occur in the next time interval δ. This number, if larger than a certain threshold, can cause that the system will alert the driver about the danger, or perhaps even apply some active safety measurements.

7.4.2 Calculation of pedestrian's movement vector

In order to calculate the impact probability we first estimate the movement vector of the pedestrian. This is essential for estimating its future trajectory and for finding out if he will hit our car.

The movement vector of the pedestrian is estimated according to several last detections. As mentioned before, the SEVILLE component, having applied a simple temporal filter, passes to the predictor an ID for each detection. The detector uses this ID to look at the last 3 positions of a given pedestrian. Given 3 positions p_1, p_2 and p_3, we can estimate the movement in a relatively reliable way, since we can check if the difference $\delta_2 = p_3 - p_2$ is similar to $\delta_1 = p_2 - p_1$. If it isn't, it implies that the movement vector is not reliable, in which case we do not estimate any impact probability for this pedestrian. If it is, then we take the average of δ_1 and δ_2 and we have a reliable movement vector of the pedestrian.

7.4.3 Impact prediction mechanism

Once we have a good positioning of a pedestrian and a good estimation of its movement vector, we have to predict if within δ seconds he will hit our car. Given the location and movement of the pedestrian and of the car, this seems like a simple geometric task to solve. However, there are certain factors which intervene :

1 The detection of the pedestrian has an estimation error. We saw in the previous subsection that if the error in the movement vector estimation is too high, we are not trying to estimate an impact probability. However, even if the error is sufficiently small, we have to take it into account.

2 We have to ask ourselves is there a possibility that the pedestrian will react in δ seconds.

3 Same for our driver : will he react in δ seconds ?

The answer for 3 is definitely negative. It is our assumption that our vehicle will not be able to change its trajectory or slow down in δ seconds, taking into account the driver and the vehicle's reaction time.

For 2, one can try to build a pedestrian model (see next subsection). However, from experiments we made it seems that the best is to assume that a pedestrian will not change its movement vector within the δ seconds in question.

The only thing which should be taken into account, therefore, is the detection error. The error σ_x, σ_y is assumed to be gaussian on world coordinates. Therefore, the prediction algorithm works as follows :

- Let $\hat{dx}$ be the speed of our vehicle (in the X axis).
- For each pedestrian with location $\langle x_0, y_0 \rangle$ and visible movement $\langle dx, dy \rangle$, do :
  - Calculate the real movement of the pedestrian $\langle dx + \hat{dx}, dy \rangle$;
  - Calculate the most likely position of the pedestrian in δ seconds $\langle x_1, y_1 \rangle = \langle x_0 + 0.3(dx + \hat{dx}), y_0 + 0.3dy \rangle$;
  - According to the real movement and the noise, draw 1000 samples from the distribution $p(\langle x, y \rangle) = e^{-\frac{(x-x_1)^2}{\sigma_x^2} - \frac{(y-y_1)^2}{\sigma_y^2}}$;

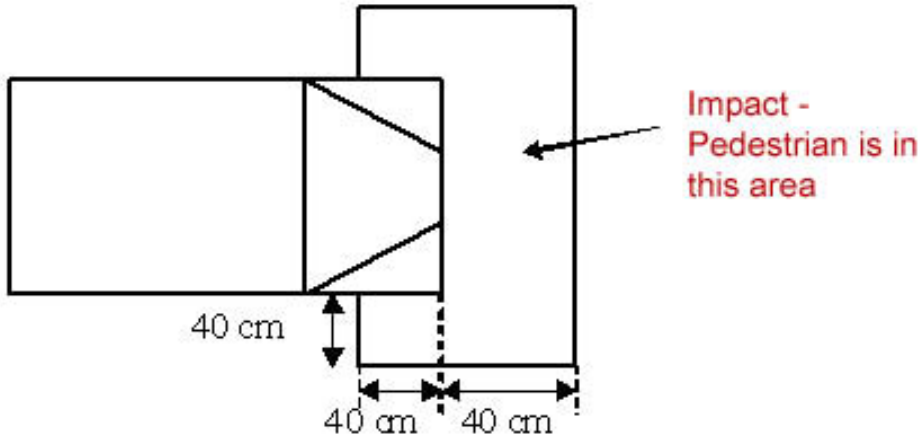


FIG. 7.16 – The zone of pedestrian impact.

– The final output probability is the number of samples which fall into the vehicle area as shown in figure 7.16, divided par 1000.

7.4.4 Using a pedestrian model

In the previous section, it is assumed that the pedestrian cannot react within δ seconds. If we assume that he can react within this time frame, we have to develop a model to predict the behavior of the pedestrian. Such a model is brought in this subsection.

The model presented here describes, how will the pedestrian behave in the next δ seconds. The model should be constructive - that is, we should be able to write a program according to that model, a program that takes a pdf of a location of a pedestrian (and optionally an approximate speed) and outputs a pdf of the location of the pedestrian after δ seconds.

The road-pavement model

To describe a reliable pedestrian model we define an area of the road in which pedestrian behaves differently. Our coordination system is centered on the front middle bottom of our vehicle, while X is oriented forwards, Y is oriented left, and Z is oriented up, all as described in Figure 7.17.

Pedestrian speed

The speed of a pedestrian is represented as a state belonging to one of the states described in table 7.1.

The motivation for expressing the speed in a discrete system is the fact that people do not move in any arbitrary speed, but in separate, discrete activities. Indeed, inside any state

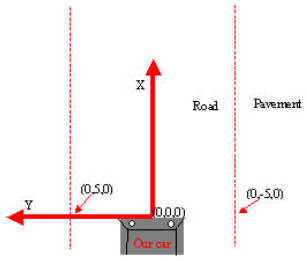


FIG. 7.17 – The axes system and the road dimensions. This is a look on the road from above.

TAB. 7.1 – The speed table of the pedestrian model.

State	Base speed (m/s)
Standing still	0
Walking	1
Jogging	2
Running	4

there is a range of continuous speeds, and we will now show how we model the continuous speeds.

A pedestrian in a speed state S is assigned a speed according to the normal distribution, where the average value is the base speed of S and the variance is 0.2 m/s.

We will now try to estimate the probability that a pedestrian will change his status in a time frame of 40 ms (typical video image timeframe). Table 7.2 contains our estimation of these probabilities for pedestrians which are on the road.

In table 7.3 we bring our estimation of these probabilities for pedestrians which are off the road. Note that an off-road pedestrian is much more likely to stop.

TAB. 7.2 – State transition probabilities for on-road pedestrians.

State	Standing	Walking	Jogging	Running
Standing	0.0001	0.99	0.009	0.0009
Walking	0.0001	0.99	0.009	0.0009
Jogging	0.0001	0.009	0.99	0.0009
Running	0.0001	0.0099	0.01	0.98

TAB. 7.3 – State transition probabilities for off-road pedestrians.

State	Standing	Walking	Jogging	Running
Standing	0.995	0.005	0	0
Walking	0.005	0.992	0.003	0
Jogging	0.005	0.003	0.99	0.002
Running	0.005	0.003	0.002	0.99

TAB. 7.4 – The pedestrian direction states, in degrees.

State	Base angle (0 is forward, to X)
Ahead	0
Back	180
Crossing left	90
Crossing right	270

Pedestrian direction

The direction of a pedestrian is, again, classified into 4 discrete states. The direction of a pedestrian is represented as a state belonging to one of the states described in table 7.4.

The motivation for expressing the direction in a discrete system is the fact that in the area of a road, people do not move in any arbitrary direction, but in separate, discrete activities. Indeed, inside any state there is a range of continuous directions, and we will now show how we model the continuous directions.

A pedestrian in a direction state S is assigned a direction angle according to the normal distribution, where the average value is the base direction of S and the divergence is 18 degrees.

We will now try to estimate the probability that a pedestrian will change his direction status in a time frame of 40 ms (typical video image timeframe). Table 7.5 contains our estimation of these probabilities for pedestrians which are on the road. These estimations are based on observations of video sequences.

In table 7.6 we bring our estimation of these probabilities for pedestrians which are off the road, on its left side.

In table 7.7 we bring our estimation of these probabilities for pedestrians which are off the road, on its right side.

TAB. 7.5 – Direction state transition probabilities for on-road pedestrians.

State	Ahead	Back	Cross Left	Cross Right
Ahead	0	0	0.5	0.5
Back	0	0	0.0005	0.9995
Cross Left	0	0	0.9995	0.0005
Cross Right	0	0	0.0005	0.9995

TAB. 7.6 – Direction state transition probabilities for off-road pedestrians, left of the road.

State	Ahead	Back	Cross Left	Cross Right
Ahead	0.99925	0.0005	0.00005	0.0002
Back	0.0005	0.99925	0.00005	0.0002
Cross Left	0.04975	0.04975	0.9	0.0005
Cross Right	0.0005	0.0005	0.0005	0.9985

TAB. 7.7 – Direction state transition probabilities for off-road pedestrians, on the right side.

State	Ahead	Back	Cross Left	Cross Right
Ahead	0.99925	0.0005	0.0002	0.00005
Back	0.0005	0.99925	0.0002	0.00005
Cross Left	0.0005	0.0005	0.9985	0.0005
Cross Right	0.04975	0.04975	0.0005	0.9

The pedestrian model function

We have written a function that simulates the movement of a random pedestrian according to the rules above. The program runs, therefore, as follows :

- Given a pedestrian in position $\langle x, y \rangle$ with speed $\langle dx, dy \rangle$, determine a speed state according to the estimated speed. If no speed information is available, start with status "standing" (speed 0).
- Determine if the pedestrian is on or off the road, and if off, on which side of the road (according to its X value).
- Calculate a new speed state according to the current state and tables 7.2 or 7.3 above (depending if the pedestrian is off- or on-road). Calculate a new direction state according to the current state and tables 7.5, 7.6, or 7.7 above (depending again on the location of the pedestrian).
- Determine the exact speed and direction angle of the pedestrian according to his states.
- Advance the pedestrian (change its location) according to his speed and direction.

The procedure above is an evolvement of a single pedestrian in a single time unit of 40 ms.

We have implemented and run a simple program which initializes 1000 pedestrians and run them simultaneously. The results of this application indeed looks like a road in the middle of the day.

7.5 Results

7.5.1 Results on several examples

A simple example

In Figure 7.18, we see a debug image with the results of several algorithms on the input scene. The pedestrian on the right was well detected by the motion segmentation, as well as by some diagonals that his legs were "spreading" along his trajectory. Some of these diagonals where, indeed, intersecting, and this allowed the legs detection algorithm to contribute its part.

Multiple pedestrians

In Figure 7.19, we see a far pedestrian and a close one. This was an experimental run which was made without the vertical edges algorithm. Moreover, the legs detection algorithm did not function so well in this sequence. However, the motion segmentation catched well both the far and the close pedestrian, while the body detection gave an exact positioning to both of them, as can be seen from the scene image, and from the likelihood image drawn from above.

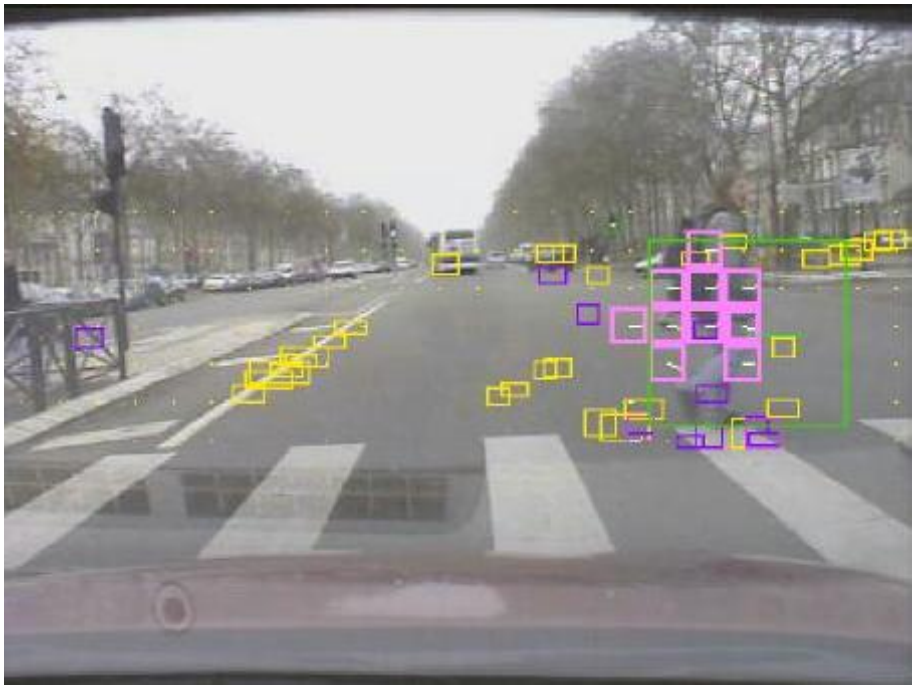


FIG. 7.18 – The simple example.

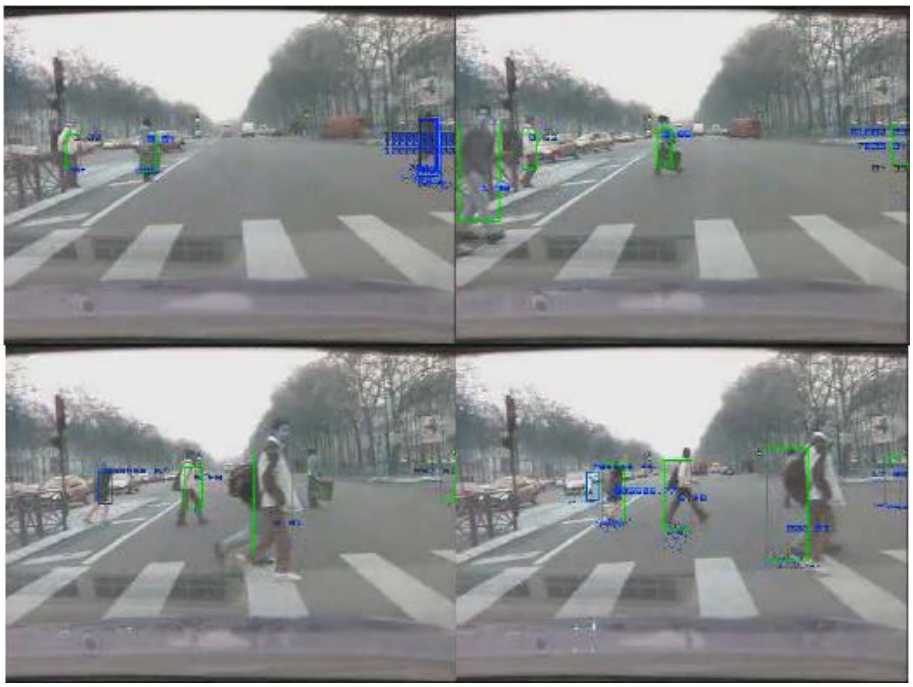


FIG. 7.19 – Several pedestrians crossing the roads, correctly detected by our pedestrian detection algorithm.

FIG. 7.20 – A pedestrian on a complex background.

Complex background

In Figure 7.20, we see a pedestrian walking on a complex background, which contains buildings, traffic signs, street lights etc. In this part of the sequence the car is turning, and motion detection becomes hard. Edges detection is not contributing much, and also body detection is, to some extent, malfunctioning because of the background. As seen in the figure, in this case the legs detection is functioning well and giving a strong positioning of the pedestrian. This is another example of the process where several algorithms work together, each time another algorithm plays the role of the "leading opinion" where the others are used only to support.

7.5.2 Summarized results

Table 7.8 summarizes the results obtained with the first set of sequences provided by Renault. On 10 scenarios, only 2 false detections were noted, and one pedestrian was not detected. Concerning the impact prediction, 2 predictions are not correct : in scenario 6, where the impact prediction is low compared to the real impact information, and in scenario 16, where no prediction is given.

7.6 Conclusion

In this chapter, we have presented an application of pedestrian detection and impact prediction. The system detects, in real-time, the existence of a pedestrian near our car and

TAB. 7.8 – Results on some test sequences, including impact prediction.

Scenario	Duration	Real impact	Impact prediction	False detections	Remarks
1	5.493s	No	No	0	Pedestrian is not detected
2	5.493s	No	No	0	OK
3	4.971s	No	No	0	OK
6	4.971s	Yes	0.8%	0	
9	4.986s	No ?	3%	0	Almost impact
16	4.986s	Yes ?	No	0	Very hard sequence
32	5.471s	No	No	0	OK
5	7.484s	Yes	2.6%	0	
7	7.484s	No	No	1	Part of car is detected
11	8.491s	No ?	1.7%	1	Almost impact

calculates the probability of collision with him.

It will be noted that for an operational system which automatically applies active measurements one needs a highly reliable system. In general it can be said that with a detection rate of 90% we can tolerate the following false detection rate :

1 For "polite" driver alert (a red light with some sound) - one false detection per 10 driving hours.

2 For brakes automatic application - one false detection per 6 months.

3 For "pedestrian airbag" or any other replaceable hardware - virtually no false detection.

On a frame rate of 25 fps with 384x288 images, even level 1 above implies one false detection out of every 180 billion sub-windows examines (based on 200,000 sub-windows per image). This rate is far better than any published results, which is one false detection every 400,000 sub-windows examined [Viol 03], or every two images.

If considering that the temporal filter can reduce this rate to one false detection out of 100 images (or 4 seconds), and the fact that only 1 out of 100 false detections will cause a driver alert (most false detections are far and/or on the sides), we still have one system fault every 400 seconds, or about 6 minutes. It is therefore clear that some improvements should be done until such a system could become a commercial product.

Chapitre 8

Conclusion

Dans cette thèse, nous avons présenté deux applications dans le domaine des transports intelligent. Après avoir passé en revue le contexte général de tels systèmes, nous avons montré pourquoi nous avons choisi de mettre en œuvre ces applications spécifiques. Puis, nous avons présenté les raisons qui nous ont amenés à choisir les méthodes appliquées. Au niveau pratique, nous avons obtenu des résultats théoriques intéressants dans divers domaines de la vision artificielle, que nous récapitulons dans la section 8.1. La section 8.2 conclut sur les applications elles-mêmes, puis terminons sur les perspectives de développement (section 8.3).

8.1 Vision artificielle

Algorithmes basés sur AdaBoost

Suite à l'analyse de l'existant en matière de détection de voitures et de piétons, nous avons pu observer que les résultats obtenus par ailleurs n'étaient pas suffisamment bons en vue d'une utilisation industrielle devant fonctionner en temps réel et ne tolérant que très peu d'erreurs.

Nous avons choisi de nous concentrer sur les systèmes à base de AdaBoost en raison de leur efficacité et de l'existence de preuves théoriques quant à leur capacité d'apprentissage. Nous avons développé deux types de primitives visuelles pour travailler avec AdaBoost.

Les premières primitives visuelles développées sont les "points de contrôle ". Nous avons prouvé que cette primitives sont plus rapides et donnent de meilleurs résultats que l'état de l'art en prenant en compte le temps d'exécution (courbe 3D ROC). Ces primitives n'ont par ailleurs pas besoin de normalisation de l'image et consomment beaucoup moins de mémoire.

La deuxième primitive visuelle développée a été adaptée aux besoins du projet CAMELLIA, où le matériel spécialisé développé a permis d'accélérer les opérations de vision.

Nous avons examiné ces nouvelles primitives aussi bien sur des données synthétiques que sur des données réelles, en les comparant aux primitives de référence. Nous avons prouvé que nos primitives sont meilleures.

Nous avons présenté quelques résultats intéressants au sujet de la "cascade" présentée par Viola et Jones. Nous avons montré que le taux de détection est meilleur dans le cas de détecteurs non-cascadés, et ne peut être atteint par des détecteurs cascadés. Cependant, si

l'on tient compte du facteur temps (courbe 3D ROC), la mise en cascade prend tout son sens et offre des performances très supérieures.

Une dernière expérience a été tirée sur l'utilisation de différents types de cameras. Le résultat principal est que, de nuit, une caméra à infrarouge lointain (*FIR - Far Infra Red*) donne des résultats de détection sensiblement supérieurs.

Evaluation du mouvement

Nous avons mis en œuvre des algorithmes développés par Wittebrood et de Haan [Witt 01]. Ces algorithmes sont très efficaces et détectent très fidèlement le mouvement. Le but original de ces algorithmes étant la compression d'image, ils ne sont pas particulièrement adaptés à la détection d'objet. Puisque nous nous intéressons à la localisation et à la vitesse des objets mobiles, nous avons adapté ces algorithmes à nos besoins, tout en développant l'idée d'unicité *(Uniqueness)*. Nous avons prouvé que l'unicité peut être un bon outil pour améliorer la détection d'objets.

Filtrage Particulaire

Nous avons montré que des algorithmes de vision élémentaires pris séparément ne sont pas assez efficaces pour développer un système qui soit capable de détecter des voitures ou des piétons de manière efficace. Nous avons choisi de développer un filtre particulaire pour fusionner les résultats fournis par chacun des algorithmes élémentaires et ainsi décupler la capacité de détection de l'ensemble.

Nous avons apporté une vue d'ensemble complète des algorithmes de filtrage particulaire et avons développé le code nécessaire à son intégration dans un véhicule.

8.2 Applications dans le domaine de l'automobile

Les concepts théoriques que nous avons développés ont été exploités dans deux applications du domaine des transports intelligents, l'ACC et la prédiction de collision d'un piéton.

L'application ACC "Stop&Go" emploie le filtrage particulaire pour suivre les véhicules précédant notre voiture. Nous avons montré comment divers algorithmes traditionnels sont combinés pour obtenir des résultats fiables.

La prédiction de collision d'un piéton a été menée en utilisant deux méthodes. La première méthode applique un détecteur d'AdaBoost sur l'image entière, et utilise les détections résultantes. La seconde est plus avancée et utilise le filtrage particulaire pour limiter le domaine de recherche.

Les deux versions de l'application dédiée aux piétons utilisent un modèle de prévision de la probabilité de choc. Nous avons donné les détails de ce modèle, basé sur une analyse du comportement des piétons.

8.3 Perspectives

Plusieurs directions existent pour continuer le travail engagé au cours de cette thèse. Dans le domaine des applications, on peut continuer et augmenter la robustesse des deux applications, en faisant plus d'essais et en analysant les problèmes rencontrés. Pour l'application de piétons, il est clair que des améliorations du taux de détection sont nécessaires pour qu'un tel système puisse aboutir à un produit commercial.

Concernant l'ACC, l'application pourrait certainement employer plus d'algorithmes dans le cadre du filtrage particulaire.

Dans le domaine plus théorique de la vision, la recherche peut être prolongée dans plusieurs directions décrites à la fin du chapitre 3. Ces directions profiteront à la détection d'objets en général, et pas simplement dans le domaine de transport, tant l'exploitation d'AdaBoost et de ses variantes apparaît riche de promesses.

* * * * *

In this thesis, we have presented two intelligent-vehicle applications. After reviewing the general context of intelligent transportation systems, we have shown why we chose to implement these specific applications. Then, we presented the reasons that led us to choose the specific ways to implement the applications.

On the way to the ready applications, we obtained some interesting theoretical results in various sub-domains of computer vision. In the section 8.4 we conclude these results. The conclusions related to the specific applications are brought later, in section 8.5.

8.4 Computer vision

AdaBoost-based algorithms

We have reviewed previous results in car and pedestrian detection. We saw, that these results were not sufficient for real-world applications that should be running on real-time and obtaining high detection rates.

We chose to concentrate on AdaBoost-based systems because of their efficiency and the existence of theoretical proofs for their detection rates. We developed two kinds of visual features to work with AdaBoost.

The first visual feature is the control-points feature. We have shown that these features are faster and give better results when tested using the time domain (3D ROC curves). These features do not need normalization of the image and consume much less memory.

The second visual feature was adapted to the needs of the CAMELLIA project, where specialized hardware was developed in order to accelerate vision operations.

We have tested the new features on synthetical as well as real-world data, comparing them with traditional, previously-presented features. We have shown that our features are better.

We have presented some interesting results concerning the attentional cascade presented by Viola and Jones. We have shown that the absolute result is better in non-cascaded detectors, but when considering the time domain, the cascade is much better.

One last experiment was done using different kinds of cameras. The main result was that at night time, far infra red (FIR) camera gives superior results.

Motion estimation

We have built on the algorithms developed by Wittebrood and de Haan [Witt 01]. These algorithms are highly efficient and detect movement.

The original goal of the developers of these algorithms was image compression, hence they were not interested in object detection. Since we needed to know the localization and speed of moving objects, we adapted these algorithms to our needs, while developing the idea of *uniqueness*. We have shown that uniqueness can be a good tool to enhance object detection.

Particle filtering

We have shown, that basic algorithms are not enough to develop a system that should track cars or pedestrians. In order to combine everything, we chose particles filter. We brought a comprehensive overview of particle filtering and developed all the code needed to integrate the other algorithms inside it.

8.5 Automotive applications

The theoretical grounds that we developed were used to develop two automotive applications, namely Stop&Go ACC and pedestrian impact prediction.

The Stop&Go ACC application is using, again, particles filter to track the vehicles in front of our car. We have shown how various traditional algorithms are combined to achieve reliable results.

The pedestrian impact prediction was presented using two methods. The first method is applying an AdaBoost detector on the entire image, and using the resulting detections. The second one is more advanced, it is using a particles filter to limit the search, thus running in real time.

We have shown, that both versions of the pedestrian applications use a special model to predict the impact probability. We have given the details of this model, based on pedestrian behavior.

8.6 Future work

Several directions exist to continue the work of this thesis. In the application domain, one can continue in making the two application more robust. This involves more testing and analyzing the problems that occur.

For the pedestrian detection application, it is clear that some improvements in the detection rate should be done until such a system could become a commercial product. For the Stop&Go, the application can perhaps use more algorithms in the framework of the particle filter.

In the theoretical vision domain, the research can be extended to several directions, as described in the end of chapter 3. These directions will improve the general domain of object detection, not just in the transportation domain.

Références bibliographiques

[Abra 04] Y. Abramson and B. Steux. "CAMELLIA report D3.2". Tech. Rep., European IST-2001-34410 CAMELLIA project, Feb. 2004.

[Amit 97] Y. Amit, D. Geman, and K. Wilder. "Joint induction of shape features and tree classifiers". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 22, pp. 1300–1305, 1997.

[Arul 02] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. "A Tutorial on Particle Filters for On-line Non-linear/Non-Gaussian Bayesian Tracking". *IEEE Transactions on Signal Processing*, Vol. 50, No. 2, pp. 174–188, Feb. 2002.

[Berg 99] N. Bergman. *Recursive Bayesian estimation : Navigation and tracking applications*. PhD thesis, Linköping University, Linköping, Sweden, 1999.

[Bert 00] M. Bertozzi, A. Broggi, A. Fascioli, and M. Sechi. "Shaped-based Pedestrian detection". In : *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 215–220, Dearbon (MI) USA, Oct. 2000.

[Bert 02] M. Bertozzi, A. Broggi, A. Fascioli, and P. Lombardi. "Vision-based Pedestrian Detection : will Ants Help ?". In : *Proceedings of the IEEE Intelligent Vehicles Symposium*, Versailles, France, June 2002.

[Bish] R. Bishop. ""Whatever Happened to Automated Highway Systems (AHS) ?"". http ://faculty.washington.edu/jbs/itrans/bishopahs.htm.

[Burg 98] C. J. C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". *Data Min. Knowl. Discov.*, Vol. 2, No. 2, pp. 121–167, 1998.

[Cali] *California PATH Project website*. http ://www.path.berkeley.edu/.

[Carl 92] B. P. Carlin, N. G. Polson, and D. S. Stoffer. "A Monte Carlo approach to nonnormal and nonlinear state-space modeling". *J. Amer. Statist. Assoc.*, Vol. 87, No. 418, pp. 493–500, 1992.

[Carp 99] J. Carpenter, P. Clifford, and P. Fearnhead. "Improved particle filter for nonlinear problems". In : *Proc. Inst. Elect. Eng., Radar, Sonar, Navig.*, pp. 2–7, Feb. 1999.

[Cris 99] D. Crisan, P. D. Moral, , and T. J. Lyons. "Non-linear filtering using branching and interacting particle systems". *Markov Processes Related Fields*, Vol. 5, No. 3, pp. 293–319, 1999.

[Douc 01] A. Doucet, S. Godsill, and C. Andrieu. "On sequential Monte Carlo sampling methods for Bayesian filtering". *Statist. Comput.*, Vol. 10, No. 3, pp. 197–208, 2001.

[Douc 02] A. Doucet, N. de Freitas, and N. Gordon, Eds. *Sequential Monte Carlo Methods in Practice. Statistics for Engineering and Information Science*, Springer-Verlag, New York Berlin Heidelberg, 2002.

[Druc 93] H. Drucker, R. Schapire, and P. Simard. "Boosting Performance in Neural Networks". *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 7, No. 4, pp. 705–719, 1993.

[Evgc 00] T. Evgcniou, M. Pontil, C. Papagcorgiou, , and T. Poggio. "Image representations for object detection using kernel classifiers". In : *Asian Conference on Computer Vision*, pp. 687–692, 2000.

[Fink 04] M. Fink and P. Perona. "Mutual Boosting for Contextual Inference". In : S. Thrun, L. Saul, and B. Schölkopf, Eds., *Advances in Neural Information Processing Systems 16*, MIT Press, Cambridge, MA, 2004.

[Freu 90] Y. Freund. "Boosting a weak learning algorithm by majority". In : *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pp. 202–216, Aug. 1990.

[Freu 95] Y. Freund and R. E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In : *European Conference on Computational Learning Theory*, pp. 23–37, 1995.

[Freu 99] Y. Freund and R. Schapire. "A short introduction to boosting". *Journal of Japanese Society for Artificial Intelligence*, Vol. 14, No. 5, pp. 771–780, 1999.

[G Re 95] H.-H. B. und G. Reichart. "Prometheus : Vision des 'intelligenten Automobils' auf 'intelligenter Straße' ? Versuch einer kritischen Würdigung". *ATZ Automobiltechnische Zeitschrift*, Vol. 97, No. 4, pp. 200–205, 1995.

[Gavr 01] D. Gavrila. "Sensor-based pedestrian detection". *IEEE Intelligent Systems*, Vol. 16, No. 6, pp. 77–81, 2001.

[Gods 00] S. Godsill, A. Doucet, , and M. West. "Methodology for Monte Carlo smoothing with application to time-varying autoregressions". In : *Proc. Int. Symp. Frontiers Time Series Modeling*, Feb. 2000.

[Gord 93] N. Gordon, D. Salmond, and A. F. M. Smith. "Novel approach to nonlinear and non-Gaussian Bayesian state estimation". In : *Proc. Inst. Elect. Eng.*, pp. 107–113, 1993.

[Hide 96] T. Hideo. "Intelligent transportation systems in Japan". *Public Roads*, Vol. 60, No. 2, pp. 174–188, Oct. 1996.

[IHar 98] I.Haritaoglu, D.Harwood, and L.Davis. "Who, when, where, what : a real time system for detecting and tracking people". In : *Proceedings of the Third Face and Gesture Recognition Conference*, pp. 222–227, Nara, Japan, Apr. 1998.

[Itti 98] L. Itti, C. Koch, and E. Niebur. "A model of saliency-based visual attention forrapid scene analysis". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 11, pp. 1254–1259, 1998.

[Jach 03] J. Jachalsky, M. Wahle, P. Pirsch, S. Capperon, W. Gehrke, W. Kruijtzer, and A. Nuñez. "A Core for Ambient and Mobile Intelligent Imaging Applications". In : *Proceedings of the 2003 IEEE International Conference on Multimedia & Expo (ICME 2003)*, p. CDROM, July 2003.

[Jazw 70] A. H. Jazwinski, Ed. *Stochastic Processes and Filtering Theory.* New York : Academic, 1970.

[Kana 95] K. Kanazawa, D. Koller, and S. J. Russell. "Stochastic simulation algorithms for dynamic probabilistic networks". In : *Proc. Eleventh Annu. Conf. Uncertainty AI*, pp. 346–351, 1995.

[Kear 88] M. Kearns and L. G. Valiant. "Learning Boolean Formulae or Finite Automata is as Hard as Factoring". Tech. Rep. TR-14-88, Harvard University Aiken Computation Laboratory, Aug. 1988.

[Kear 89] M. Kearns and L. G. Valiant. "Cryptographic Limitations on Learning Boolean Formulae and Finite Automata". In : *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pp. 433–444, May 1989.

[Kear 94] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[Leun 87] M. Leung and Y.H.Yang. "Human Body motion segmentation in a complex scene". *Pattern Recognition*, Vol. 20, No. 1, pp. 55–64, 1987.

[Levi 03] A. Levin, P. Viola, and Y. Freund. "Unsupervised Improvement of Visual Detectors using Co-Training". In : *Proceedings of the International Conference on Computer Vision*, pp. 626–633, 2003.

[Liu 98] J. S. Liu and R. Chen. "Sequential Monte Carlo methods for dynamical systems". *J. Amer. Statist. Assoc.*, Vol. 93, pp. 1032–1044, 1998.

[MacC 99] J. MacCormick and A. Blake. "A probabilistic exclusion principle for tracking multiple objects". In : *Proc. Int. Conf. Comput. Vision*, pp. 572–578, 1999.

[Mall 89] S. Mallat. "A theory for multiresolution signal decomposition : the wavelet representation". *IEEE transactions on pattern analysis and machine intelligence*, Vol. 1, No. 7, pp. 674–93, July 1989.

[MobilEye] *MobilEye ltd. website.* http ://www.mobileye.com.

[Moha 01] A. Mohan, C. Papageorgiou, and T. Poggio. "Example-Based Object Detection in Images by Components". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 23, No. 4, pp. 349–361, 2001.

[Mora 96] P. D. Moral. "Non-linear filtering : Interacting particle solution". *Markov Processes Related Fields*, Vol. 2, No. 4, pp. 555–580, 1996.

[Oren 97] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. "Pedestrian detection using wavelet templates". June 1997.

[Osun 97] E. Osuna, R. Freund, and F. Girosi. "Training support vector machines : an application to face detection". In : *Proceedings of the IEEE conference on computer vision and pattern recognition*, San Juan, Puerto Rico, June 1997.

[Papa 98a] C. Papageorgiou, T. Evgeniou, and T. Poggio. "A trainable pedestrian detection system". In : *Procs. IEEE Intelligent Vehicles Symposium*, pp. 241–246, Oct. 1998.

[Papa 98b] C. Papageorgiou, T. Poggio, and M.Oren. "A general framework for object detection". In : *International conference on computer vision*, pp. 555–562, Jan. 1998.

[Papa 99] C. Papageorgiou and T. Poggio. "A Pattern Classification Approach to Dynamical Object Detection". In : *ICCV (2)*, pp. 1223–1228, 1999.

[Ripl 87] B. Ripley, Ed. *Stochastic Simulation.* New York : Wiley, 1987.

[Scha 89] R. E. Schapire. "The Strength of Weak Learnability". In : *30th Annual Symposium on Foundations of Computer Science*, pp. 28–33, Oct. 1989.

[Scha 97] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. "Boosting the margin : A new explanation for the effectiveness of voting methods". In : *Machine Learning : Proceedings of the Fourteenth International Conference*, pp. 322–330, 1997.

[Scha 98] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee. "Boosting the margin : A new explanation for the effectiveness of voting methods". *The Annals of Statistics*, Vol. 26, No. 5, pp. 1651–1686, October 1998.

[seville] *SEVILLE project website.* http ://caor.ensmp.fr/ãbramson/seville/.

[Steu 00] B. Steux, P. Coulombeau, and C. Laurgeau. "Maps : a framework for prototyping automotive multi-sensor applications". In : *IEEE Intelligent Vehicles Symposium*, Oct. 2000.

[Steu 02] B. Steux. "Fade : A Vehicle Detection and Tracking System Featuring Monocular Color Vision and Radar Data Fusion". In : *IEEE Intelligent Vehicles Symposium*, June 2002.

[Tieu 99] K. H. Tieu and P. Viola. "Boosting Image Database Retrieval". Tech. Rep. 1669, MIT Artificial Intelligence Laboratory, 1999.

[Tsot 95] J. Tsotsos, S. Culhane, W. Wai, Y. Lai, N. Davis, and F. Nuflo. "Modeling visual-attention via selective tuning". *Artificial Intelligence Journal*, Vol. 78, No. 1-2, pp. 507–545, 1995.

[Tsuk 85] T. Tsukiyama and Y. Shirai. "Detection of the movements of persons from a sparse sequence of TV images". *Pattern Recognition*, Vol. 18, No. 3, pp. 207–213, 1985.

[Vali 84] L. G. Valiant. "A Theory of the Learnable". *Communications of the ACM*, Vol. 27, No. 11, pp. 1134–1142, Nov. 1984.

[Vapn 95] V. N. Vapnik. *The nature of Statistical Learning Theory.* Springer-Verlag, 1995.

[Viol 01] P. Viola and M. Jones. "Rapid Object Detection using a Boosted Cascade of Simple Features". In : *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 511– 518, 2001.

[Viol 02] P. Viola and M. Jones. "Robust Real-time Object Detection". *International Journal of Computer Vision*, 2002.

[Viol 03] P. Viola, M. J. Jones, and D. Snow. "Detecting Pedestrians using Patterns of Motion and Appearance". In : *IEEE International Conference on Computer Vision*, pp. 734–741, Nice, France, Oct. 2003.

[Witt 01] R. Wittebrood and G. de Haan. "Real-Time Recursive Motion Segmentation of Video Data on a Programmable Device". In : *IEEE Transactions on Consumer Electronics*, pp. 559–567, Aug. 2001.

Index